Designing a Programming Game to Improve Children's Procedural Abstraction Skills in Scratch

ROSE, Simon <http://orcid.org/0000-0002-8165-3016>, HABGOOD, Jacob <http://orcid.org/0000-0003-4531-0507> and JAY, Tim <http://orcid.org/0000-0003-4759-9543>

Published version

ROSE, Simon, HABGOOD, Jacob and JAY, Tim (2020). Designing a Programming Game to Improve Children's Procedural Abstraction Skills in Scratch. Journal of Educational Computing Research, 073563312093287-073563312093287.

Designing a Programming Game to Improve Children's Procedural Abstraction Skills in Scratch

Abstract:

The recent shift in compulsory education from ICT-focused computing curricula to informatics, digital literacy and computer science, has resulted in children being taught computing using block-based programming tools such as Scratch, with teaching that is often limited by school resources and teacher expertise. Even without these limitations, Scratch users often produce code with 'code smells' such as duplicate blocks and long scripts that can impact how they understand and debug projects. These code smells can be removed using procedural abstraction, an important concept in computer science that is rarely taught to this age group. This article describes the design of a novel educational block-based programming game, Pirate Plunder, to teach this concept, concentrating on how procedural abstraction is introduced and reinforced. It then reports an extended evaluation to measure the game's efficacy with children aged 10 and 11, finding that children who played the game were then able to use procedural abstraction in Scratch. The article then uses game analytics to explore why the game was effective and gives three recommendations for educational game design based on this research: using learning trajectories and restrictive success conditions to introduce complex content, increasing learner investment through customisable avatars and suggestions for improving the evaluations of educational games.

Keywords: block-based programming, game-based learning, game design, Pirate Plunder, procedural abstraction, computer science education, computational thinking

1. Introduction

This article describes the design and evaluation of a novel educational programming game, Pirate Plunder, to teach procedural abstraction to children. Pirate Plunder aims to teach children to identify code duplication in Scratch projects and be able to remove this using procedural abstraction and code reuse. Despite these skills being an important part of computer science, children, particularly those in primary education (age 5 to 11), are rarely taught them because of a lack of teacher expertise (Rich et al., 2019) and school resources (Larke, 2019). The article builds upon a previous report on the design of Pirate Plunder (Rose et al., 2018) by describing in more detail how the game introduces the learning content. It provides a complete and final report of an extended evaluation described in Rose et al. (2019). As such, it includes a detailed description of the complete study (employing a crossover design) and a comprehensive discussion of the complete set of results and their implications.

> RQ1 – Can a game-based learning approach be used to teach primary school children to use procedural abstraction in Scratch projects?
> RQ2 – What aspects of the game design influence the effectiveness of this approach and why?

This section gives an overview of the context behind these research questions, including programming tools in computer science education, code smells and procedural abstraction, before discussing similar work and how Pirate Plunder has been designed to introduce these concepts.

1.1 Computer science education

Over the last few years, ICT-focused computing curricula in compulsory education have been replaced with wider topics of informatics, digital literacy and computer science (Heintz et al., 2016). This shift has aimed to give children the skills and knowledge required in an increasingly digital world, with learning content that focuses on computer programming, robotics and computational thinking. This new type of computing curriculum was introduced in England in 2013, in part due to recommendations by Livingstone & Hope (2011) and The Royal Society (2012) in reports that were critical of the old curriculum.

Computing education in English schools is still described as "patchy and fragile" (The Royal Society, 2017, p. 6), with teachers "lacking sufficient theoretical and technical knowledge of computing" (p. 54). This stems from the curriculum being introduced without adequate training programs, infrastructure, and materials. This problem is particularly pronounced in primary education where teachers are not subject specialised. It is often left to teachers to develop an understanding of the required learning content by themselves (Rich et al., 2019), meaning that computing is often left out of classroom teaching entirely (Larke, 2019) or children are given educational programming tools with little or no support.

1.1.1 Educational programming tools

There is a multitude of educational programming tools available for use in computer science education (e.g. Cooper et al., 2003; Hooshyar et al., 2016; Weintrop & Wilensky, 2013). These differ in type, cost, complexity and learning approach, meaning that teachers are often unsure which they should be using. The problem is exacerbated by the limitations in hardware and financial restraints in schools. Yet, there is definite potential for these tools to be used to support children's understanding of key concepts in computing.

These tools include visual programming environments, games, physical devices and unplugged activities. They range from symbolic drag-and-drop programming to text-based programming languages that allow procedures, variables, iteration and conditional execution (Duncan et al., 2014). Novices can find it easier to learn computer science using block-based languages, relative to text-based languages, because they rely on recognition instead of recall (blocks are selected from a pallet), reduce cognitive load by chunking code into smaller numbers of meaningful elements and allow users to avoid basic errors by providing constrained direct manipulation of structure (Bau et al., 2017).

The most widely used educational programming tool is Scratch, with over 53 million projects shared on its online platform since its public release in 2007 (Scratch Team, 2020). It is also the most popular environment in primary education (Rich et al., 2019). Scratch is a visual programming environment designed for children age 8 and above. It aims to "introduce programming to those with no previous programming experience" (Maloney et al., 2010, p. 2). Scratch uses a block-based programming language in which blocks are combined to form scripts. Its design is inspired by constructionism (Papert, 1980), a learning theory where knowledge and problem-solving skills are developed through exploration. Scratch supports constructionism by always having the block-palette visible, having little in-built guidance and feedback and no error messages. This type of block-based programming is also prevalent in other popular tools used in primary education including Code.org (Code.org, 2020), Tynker (Neuron Fuel, 2020), Hopscotch (Hopscotch Technologies, 2020) and Purple Mash (2Simple Ltd, 2020).

1.1.2 Computational thinking

It is difficult to discuss computer science education without acknowledging the idea of computational thinking and its implications. Computational thinking stems from the idea that computer science can help develop wider problem-solving and logical thinking skills. It involves skills such as working at multiple levels of abstraction, writing algorithms, understanding flow control, recognising patterns and decomposing problems (Rose et al., 2017). Proponents of computational thinking take this one step further, suggesting that these skills are a "foundational competency for every child" (Grover et al., 2018, p. 1). As such, computational thinking has been used by policymakers as justification for introducing computer science into compulsory education. Yet, it has been criticised by some for its 'decoupling' from the theoretical foundations of computer science, along with the lack of evidence for it as a multidisciplinary problem-solving skill (Denning, 2017).

Nonetheless, in this article, we concentrate on the use of these skills in computer programming, using the computational thinking measures Dr. Scratch (Moreno-León & Robles, 2015), to measure abstraction in Scratch projects, and the Computational Thinking test (Román-González et al., 2018), used as a secondary measure of improvements on the computational thinking skills used in programming.

1.2 Code smells and bad programming practices

The constructionist programming approach in block-based environments like Scratch can result in poor programming practices because proper software engineering principles are not introduced (Dorling & White, 2015). Code smells are a useful method for identifying these bad practices. The term 'code smell' was coined by Fowler (1999) and refers to a surface indication in a program that usually corresponds to a deeper problem. Identifying code smells and 'refactoring' code to remove them can help improve the design and readability of code. Refactoring is the "process of changing a software system in such a way that it does not alter the external behaviour of the code but improves its internal structure" (p. 9). Fowler gives a list of possible code smells found in object-oriented programming, including duplicated code, long methods, large classes, and long parameter lists.

1.2.1 Code Smells in Scratch

Similar code smells are also found in block-based programming languages like Scratch. As stated in Section 1.1.1, Scratch supports a constructionist programming approach, where solutions are unplanned and created largely through self-directed exploration or 'tinkering' (Maloney et al., 2010). This allows learners to quickly create programs. However, it can result in the learner producing 'bad' code (indicated by code smells) and forming bad programming

habits (Meerbaum-Salant et al., 2011). This is particularly significant because of the widespread use of Scratch in compulsory education and the lack of relevant teacher knowledge within the profession.

The most common code smells in Scratch projects are duplicated blocks (a repeated sequence of blocks, regardless of block inputs, that are used to reuse code), large scripts (a long script that could be shortened through better code reuse) and dead code (blocks that can be safely removed from the program without affecting its behaviour). Table 1 shows the code smell prevalence in four exploratory analyses of large Scratch project repositories.

Table 1: Scratch project analyses for code smells

Author(s)	Number of projects analysed	% of projects		
		Duplicated blocks	Long scripts	Dead code
Moreno-León & Robles (2014)	100	62%	*Not analysed*	*Not analysed*
Aivalaglou & Hermans (2016)	247,798	26%	30%	28%
Techapalokul (2017)	1,066,308	46%	47%	23%
Robles et al. (2017)	250,166	20%	*Not analysed*	*Not analysed*

Hermans & Aivalaglou (2016) found that code smells can impact understanding, debugging and the ease with which learners can alter projects. Furthermore, Techapalokul & Tilevich (2015) found that novice programmers that are "prone to introducing some smells continue to do so even as they gain experience" (p. 10). Code smells are particularly important because 'remixing' other users' projects is a large part of the Scratch online platform (Dasgupta et al., 2016).

1.3 Procedural abstraction and the extract method

Procedural abstraction can be used to remove code smells. It is an important computer science skill and is one of the two kinds of abstraction utilised in computer science, with the other being data abstraction (Haberman, 2004). Procedural abstraction involves the separation of the logical properties of an action from the implementation details. This is often done by moving fragments of code into a procedure (with or without data arguments to pass information to it) that can then be invoked in multiple places, a process known as the 'extract method' (Fowler, 1999). Kallia & Sentance (2017) describe procedural abstraction (and its subconcepts) as a potential 'threshold concept' in computer science. That is, a concept that opens "up a new and previously inaccessible way of thinking about something" (Meyer & Land, 2003, p. 1). However, procedural abstraction is difficult for novices and an area where misconceptions often arise.

Duplicated blocks, large scripts and dead code smells in Scratch can all be removed using procedural abstraction and the extract method. This is done using 'custom blocks' (procedures in Scratch) that the user can define. These can be given 'inputs' (parameters) to pass data to the block. For example, Figure 1 shows an example of a Scratch project that contains a duplicated code smell: the 'point in direction', 'repeat' and 'move' blocks are all repeated four times with different input values. Figure 2 shows the same project with the duplicated blocks 'extracted' into a custom block called 'turnAndMove' that takes two arguments, distance, and degrees. These projects are shown using Scratch 2, as Scratch 3 had not been released when this study was conducted. Scratch is limited in that custom blocks can only be used within the sprite they are defined, a limitation that has led to calls for a functionality change (Techapalokul & Tilevich, 2019b).

In their Scratch project analysis, Robles et al. (2017) found that the use of custom blocks and cloning (duplicating sprites at runtime) did not impact the amount of code duplication. This supports the idea that procedural abstraction is difficult for novices, particularly children, to use appropriately.

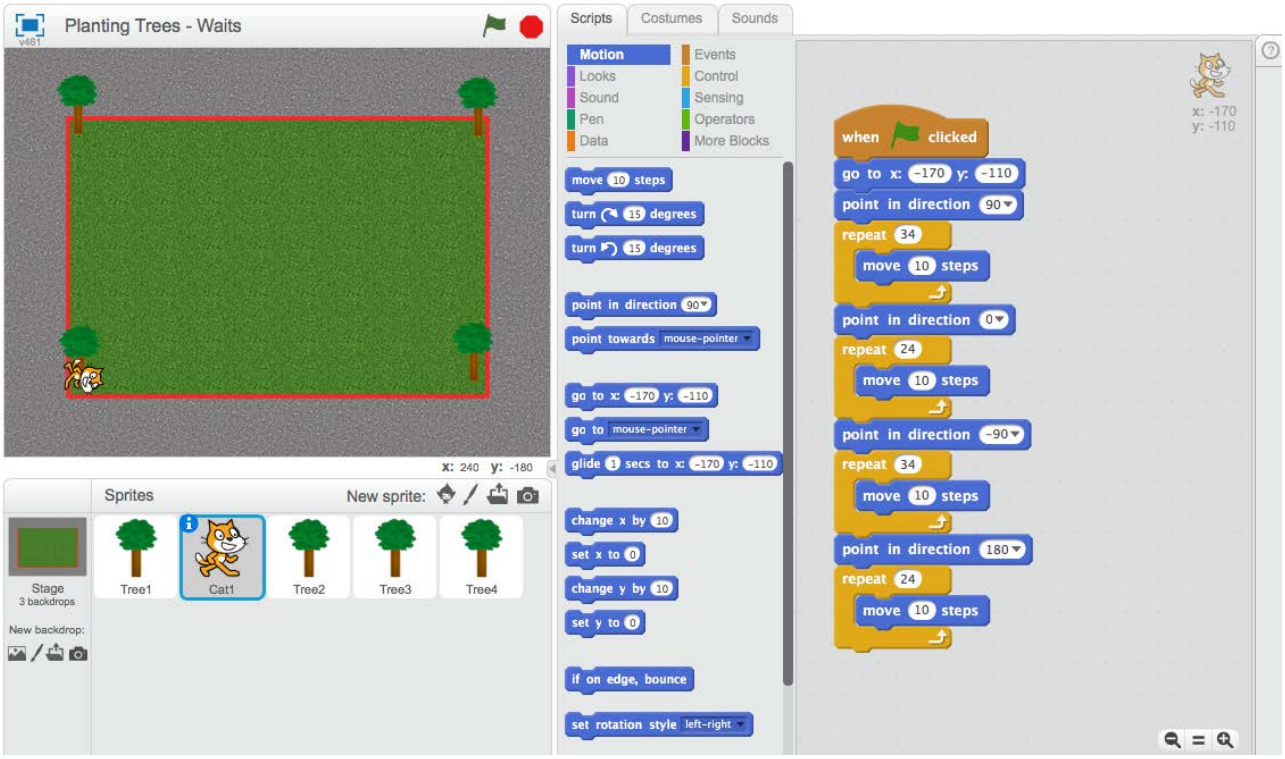

Figure 1: Moving a cat sprite around the four corners of a park using block duplication

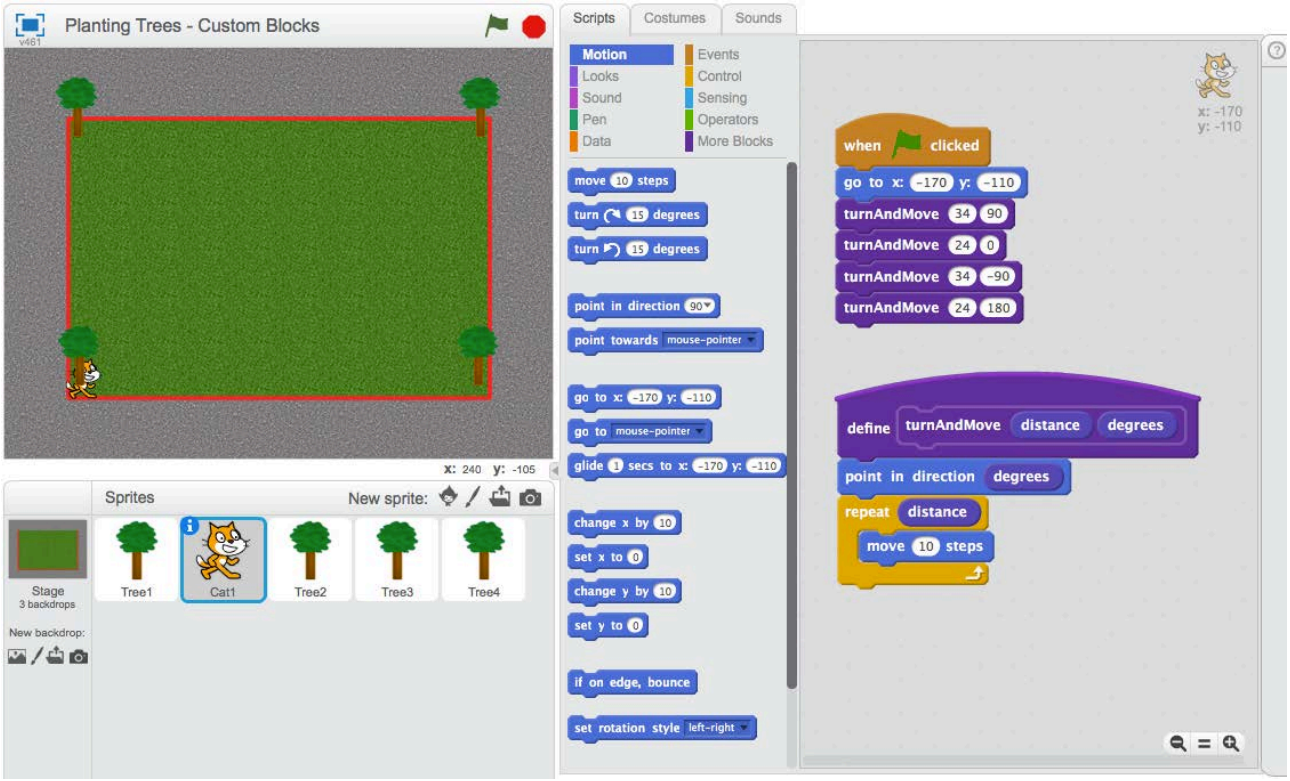

Figure 2: Using the extract method and a custom block to refactor the project in Figure 1

1.4 Game design to introduce and reinforce procedural abstraction

So far, we have highlighted that children are being taught computer science in primary education using block-based programming tools such as Scratch, with teaching that is often limited by school resources and teacher expertise. The open-ended design of these block-based tools can allow users to produce 'bad' code, which can be recognised through code smells, such as duplicated blocks and long scripts. These code smells can be refactored through the extract method, a skill that requires procedural abstraction, itself an important part of computer science.

We hypothesise that game-based learning can play a key role in supporting computer science learning in block-based visual programming environments. Procedural abstraction is a good focus for this because it is challenging for primary school children and is not part of the current English national curriculum.

1.4.1 Related work

There have been several efforts to teach the higher-level principles of abstraction to children, getting them to think about and articulate problems at different levels of abstraction (e.g. Rijke et al., 2018; Statter & Armoni, 2020). However, research on procedural abstraction in block-based programming tools is limited. Sherman & Martin (2015) use it as part of their computational thinking rubric in the App Inventor block-based tool (Wolber, 2011). Techapalokul & Tilevech (2019a) created an automated refactoring tool for Scratch that removes code smells using the extract method. Although similar, our work differs from this in that we aim to teach children to do this process themselves and to understand why this is beneficial. Kalas & Benton (2017) explored the factors that underpin primary school children's understanding of procedural abstraction and suggest a pedagogical approach that they found effective. Pirate Plunder aims to do this without teacher instruction, which leads us to our first research question (RQ1): can a game-based learning approach be used to teach primary school children to use procedural abstraction in Scratch projects?

Procedural abstraction is introduced in several existing educational games. How this is implemented depends on the type of programming interface used, which can be split into two categories: games that use symbolic programming interfaces with minimal text, including Lightbot (Yaroslavaski, 2014) and AutoThinking (Hooshyar et al., 2019), and those that use text-heavy block-based languages like Google Blockly (Google, 2020), including ctGameStudio (Werneburg et al., 2016) and Dragon Architect (Bauer et al., 2017). In the first category, procedural abstraction is implemented similarly to Lightbot, which gives the player one or two procedures that can be called from the main set of instructions (Figure 3). AutoThinking uses are similar approach, allowing the player to save sets of instructions to procedures that can then be executed using their number identifier from the main program. Whilst these are useful ways of introducing procedural abstraction, they do not allow players to use parameters to pass data to their procedures. Understanding parameters is an important part of learning to use procedures as it allows for further code reuse. In the second category, procedures are created and used from the block pallet similar to Scratch (Section 1.3). Pirate Plunder falls into this category but aims to teach procedural abstraction using a block-based syntax that is similar to Scratch, which is simpler than Google Blockly. The rest of this section explains the novel approach to introducing this content: concept scaffolding, tutorials and feedback and customisation, which leads into RQ2: what aspects of the game design influence the effectiveness of this approach and why?

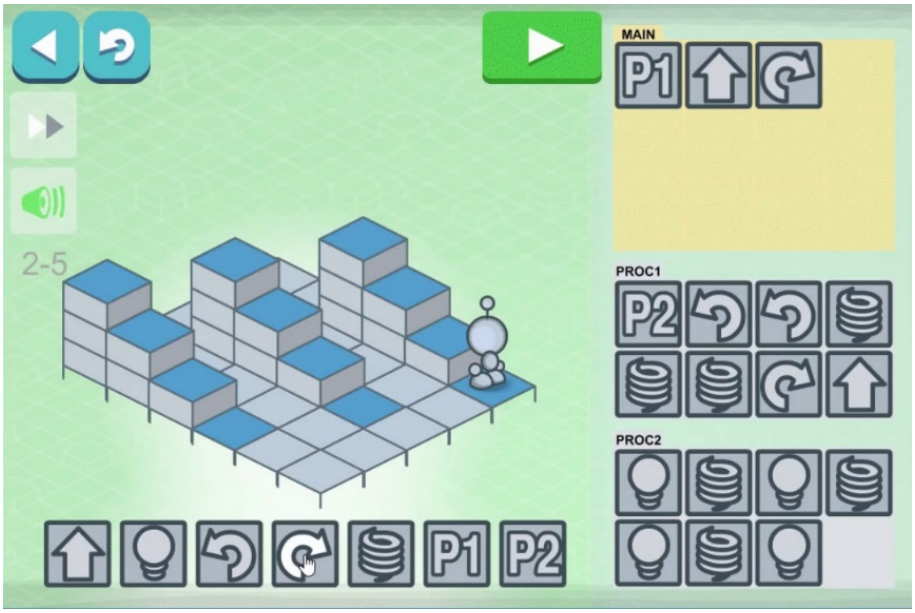

Figure 3: How Lightbot introduces procedural abstraction

1.4.2 Concept scaffolding

Pirate Plunder introduces procedural abstraction through a series of scaffolded concepts that are designed to justify the rationale behind using them. This focuses on both applied knowledge and contextualised theoretical knowledge, in line with recommendations by Kazimoglu, Kiernan, Bacon & Mackinnon (2011) when introducing computer

programming concepts in serious games. The game begins with basic Scratch blocks (events and motion) that do not require any prior Scratch knowledge. The learning content is introduced intrinsically as part of the game, an approach that Habgood & Ainsworth (2011) found to be effective in educational games.

We designed a learning trajectory for the game to scaffold the concepts in a way that children would understand (Figure 4). Learning trajectories are used in mathematics to define the developmental progression to a mathematical goal, giving a set of instructional tasks that help children develop higher levels of thinking (Clements & Sarama, 2004). The goal of Pirate Plunder is to introduce procedural abstraction. The levels are designed as instructional tasks to help the player understand the concepts along the trajectory, with the Scratch blocks or functionality for that concept only made available to the player once they have reached a certain level.

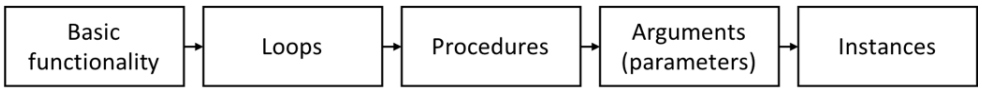

Figure 4: Pirate Plunder learning trajectory

The difficulty progression is described in more detail in previous work (Rose, 2019). In summary, the levels in each stage (or concept) get progressively more difficult, requiring more duplication and longer scripts. For each subsequent stage, the player is then introduced to a technique or block that can be used to remove that duplication, moving along the learning trajectory. For example, in the final 'loops' level, the player must use three duplicated 'repeat 5, move 1' blocks and two duplicated 'repeat 2, move 1' blocks (Figure 5). Figure 6 and Figure 7 show how this could be refactored using custom blocks (procedures) and custom blocks with two inputs (a parameterised procedure). Concept scaffolding in Pirate Plunder is particularly important because it progresses from procedures to parameterised procedures, which are one of the most challenging concepts in introductory programming (Madison & Gifford, 1997).

Figure 5: The final 'loops' level before custom blocks are introduced

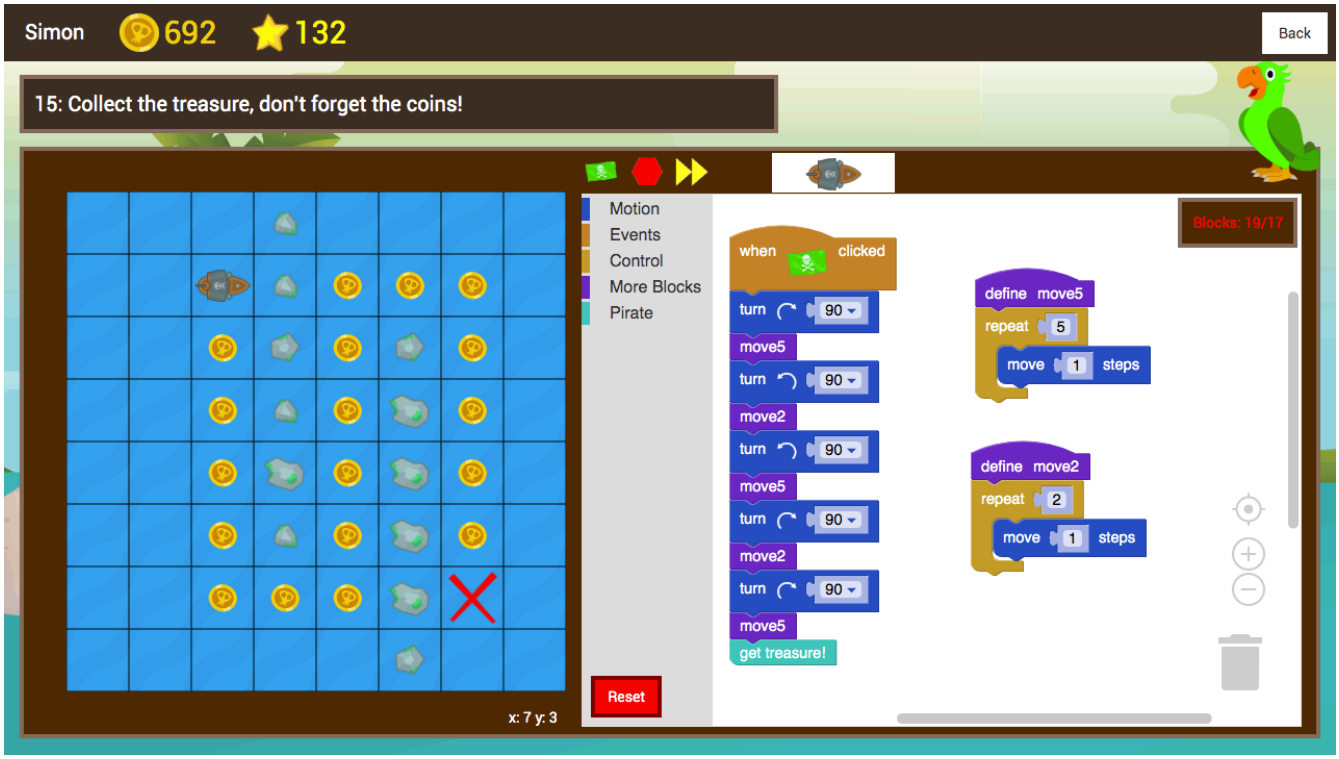

Figure 6: The level in Figure 5 completed using two procedures (no parameters)

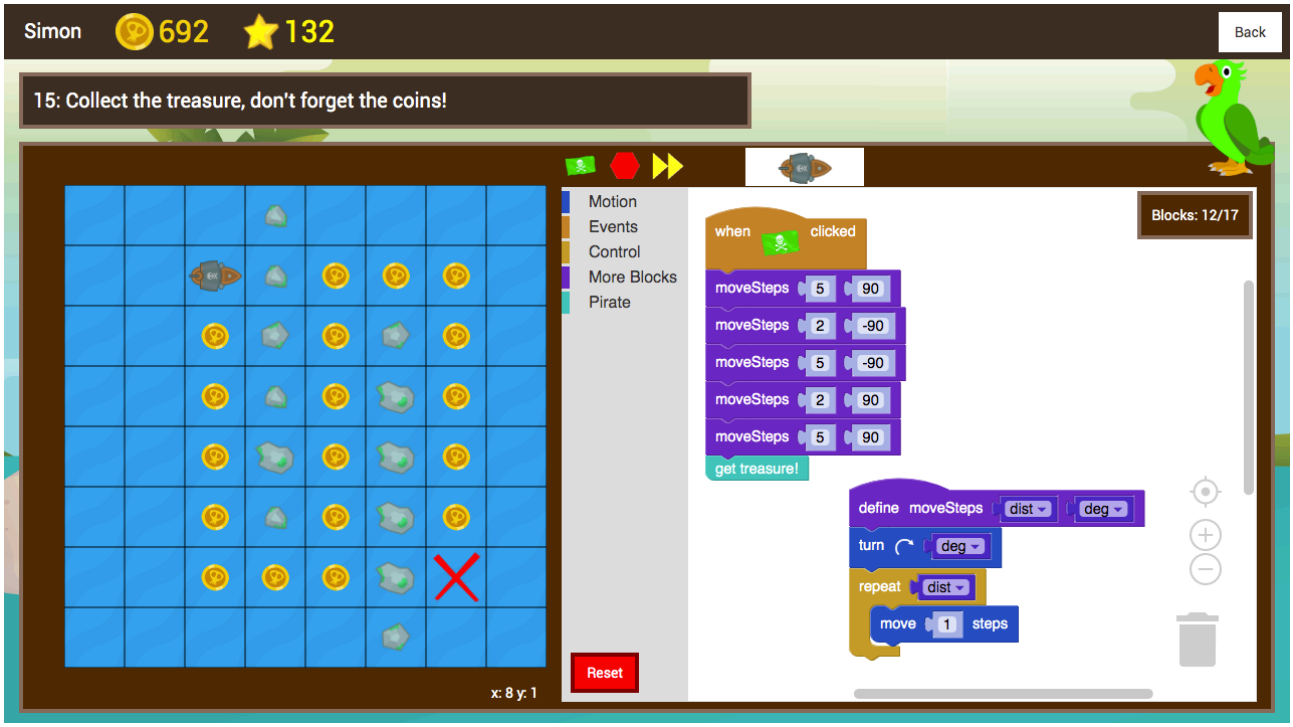

Figure 7: The level in Figure 5 completed using a parameterised procedure

The final concept in the learning trajectory is instances of sprites. In Scratch, this is done through cloning, which allows a sprite to create a clone of itself or another sprite whilst the project is running. This duplicate is a separate instance of the original or parent sprite but will inherit properties from the parent that can be modified. Cloning a sprite is similar to creating an instance of a class in object-oriented programming, where an 'instance' is a concrete occurrence of an object that is created during runtime. Although not part of procedural abstraction, it was included in the learning trajectory because Dr. Scratch uses it as a measure of abstraction (Moreno-León & Robles, 2015) and this was our primary measure during the study.

Pirate Plunder contains 40 levels that were used in the data analysis (Figure 8). There are also eight 'general' levels designed as a further challenge to players who have completed the game. These all unlock when the player completes level 40.

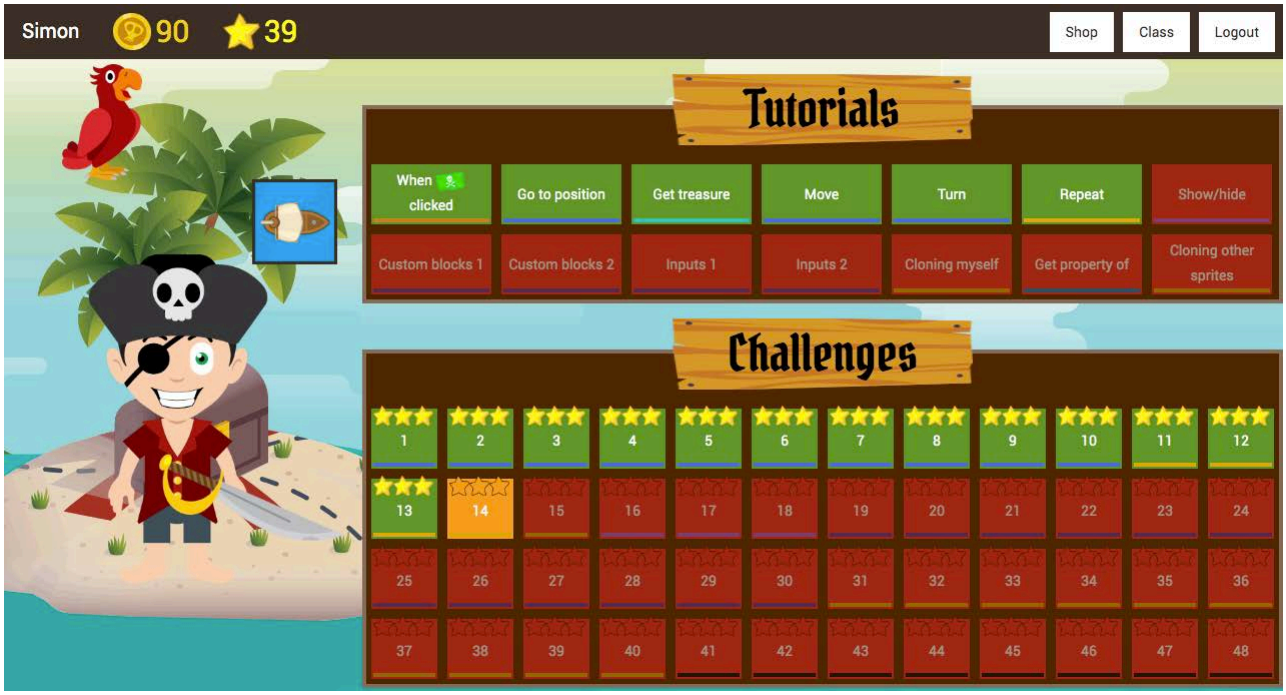

Figure 8: Pirate Plunder player avatar and level select (general levels are indicated with black bars)

The game motivates players to use the taught functionality through block limits, collectable items, required block validation and obstacles. Each challenge limits the number of total blocks that can be used in the program, forcing the player to address block duplication and produce an optimal solution.

1.4.3 Tutorials and feedback

Pirate Plunder combines 'process constraints': increasing the number of features (in this case, blocks) that the player can control as they progress through the game, with 'explanations' that specify exactly how to perform an action (Lazonder & Harmsen, 2016). These explanations are made in 'tutorial' levels (Figure 9) that introduce the player to the next concept on the learning trajectory. The player then uses these concepts or blocks in 'challenge' levels.

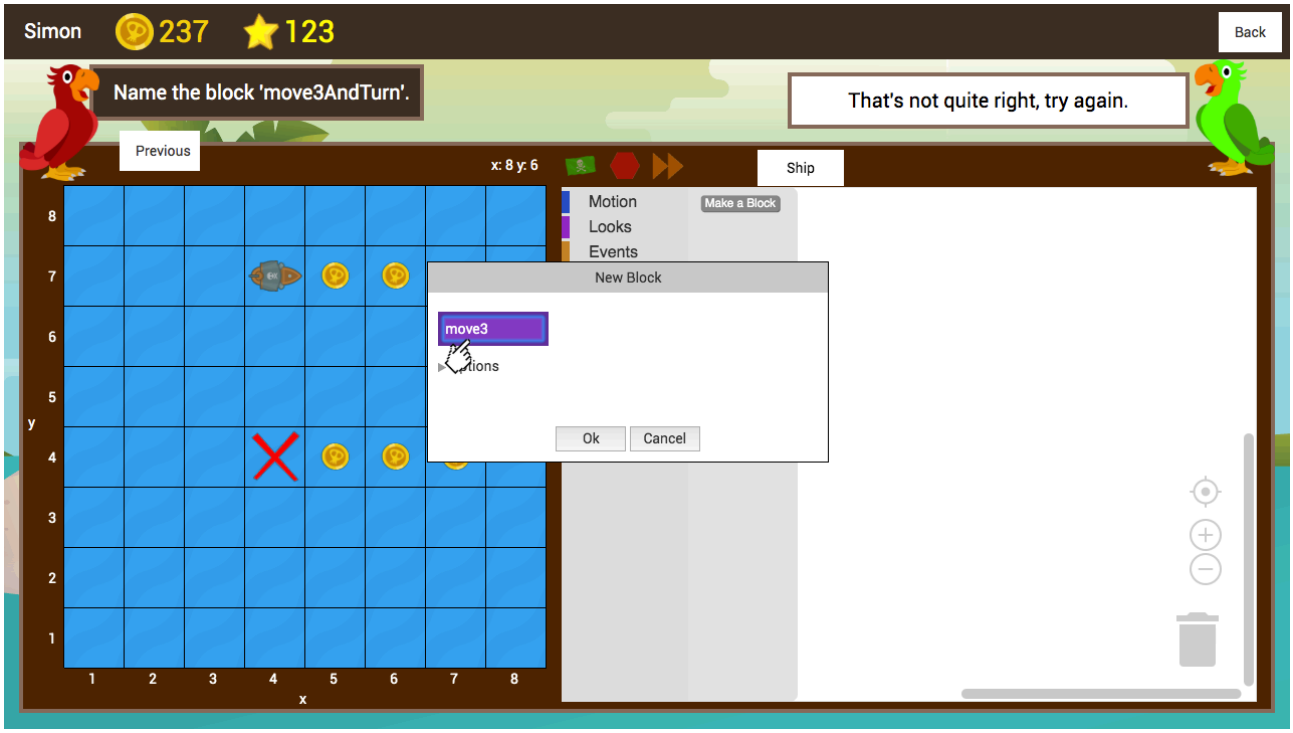

Figure 9: A stage of the 'custom blocks' tutorial level in Pirate Plunder, in which the player is guided through the tutorial by the red parrot in the top left corner and is given feedback on each stage by the green parrot in the top right

Tutorial levels guide the player through using each concept or block. This approach was inspired by the moving tutorial character from Dragon Architect (Bauer et al., 2017), a block-based programming game, and Stagecast Creator (Seals et al., 2002), a rule-based visual programming environment with a section that walks the player through the application functionality. Pirate Plunder uses a combination of these two approaches. Players must follow a series of actions within a tutorial that are highlighted on-screen by a moving help character (the red parrot). They must follow the actions correctly to complete the level. These differ from 'challenge' levels, where the player is not guided through the level by the help character. The combination of tutorials, challenges and the learning trajectory was designed as a novel way to deliver conceptually difficult content. Later tutorials show refactored challenge levels that the player has already seen, demonstrating the justification for using those blocks later in the game.

Players are given feedback on both tutorial and challenge levels by the green parrot (Figure 10). This is automatically given for guidance, warnings and level validation. Guidance feedback is for general block use and program issues, such as missing an 'event' block to start program execution. Warnings are for behaviour that might break the game (e.g. recursion). Level requirement feedback is given for not using the required blocks and reaching the block limit.

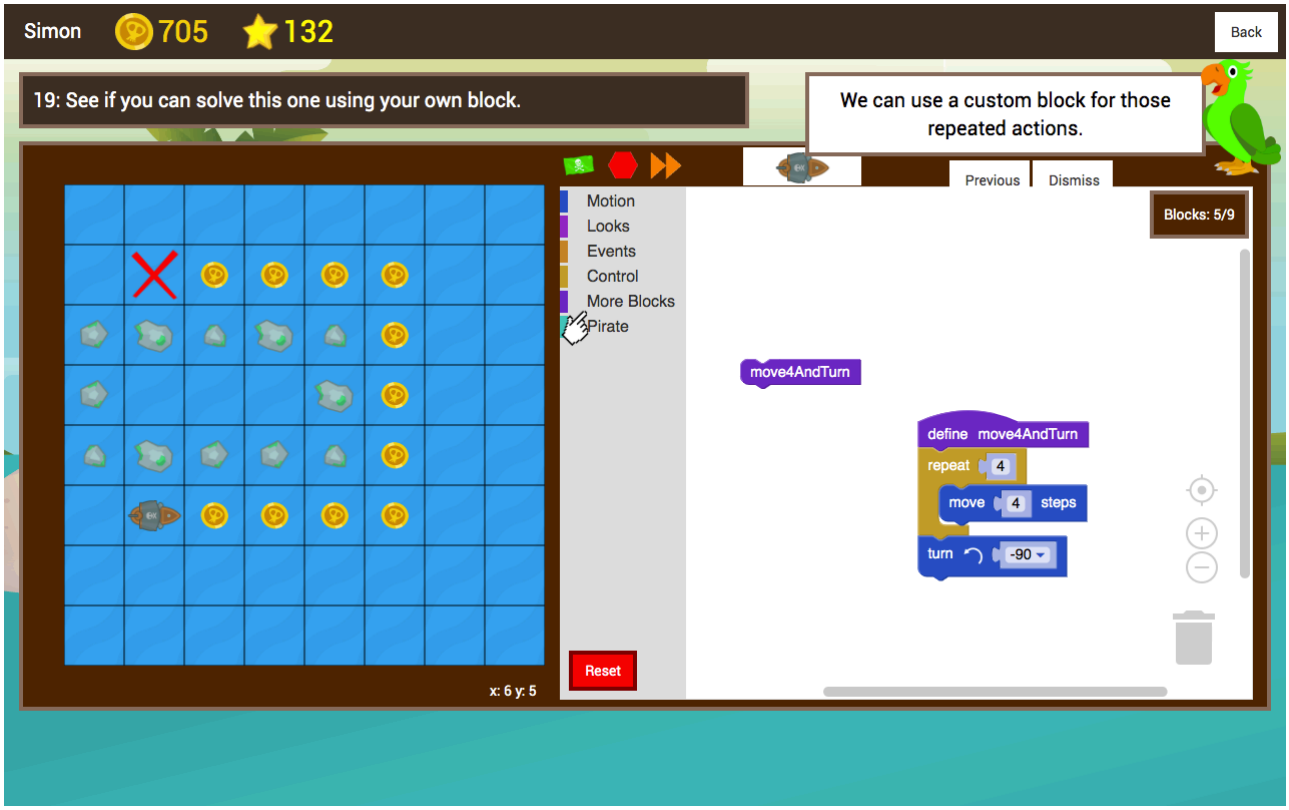

Figure 10: A Pirate Plunder challenge level, in which the green parrot gives both hints (when clicked on) and feedback

On challenge levels, the player can also ask the green parrot for hints on how to complete the level. The player is given between three and six hints designed to guide the player. Table 2 shows the hints given on the first procedures challenge.

Table 2: The hints given on the first Pirate Plunder procedures (custom blocks) challenge

Hint number	Text	Explanation
1	"Take a look at the grid, what tasks can we repeat?"	Recognising the duplicated functionality is a key part of creating the correct custom block(s) (procedures) for each level. This first suggestion is used in all of the custom block and inputs (parameters) challenges.
2	"We repeat 4 move 1 and turn left. We do this 3 times."	Building on the last hint, indirectly telling the player what should go inside the custom block.
3	"We can use a custom block for those repeated actions."	Reminding the player that they should be using custom blocks for those repeated actions.
4	"Have you added blocks to the 'define' block?" (pointer to the first define block in the program.)	This was an issue observed when players used custom blocks in an earlier study. Players would not add blocks to their define blocks in early custom block challenges. Instead, they would expect that the block would achieve the required functionality by naming the define block what they wanted it to do and not adding blocks to it.
5	"You can use your block from the 'more blocks' folder." (pointer to the 'more blocks' folder.)	Reminding the player that once they have created a custom block, they need to use it from the 'More Blocks' folder.
6	"Do you want to do the custom block tutorials again?" (option to return to the level select screen.)	Telling the player to go and do the custom blocks tutorial again if the previous suggestions did not help them complete the level.

1.4.4 Customisation

Customisable player avatars are an important part of keeping players motivated throughout Pirate Plunder. Each player has an avatar that they can purchase items for using coins collected when playing through levels Figure 8. These items unlock as the player progresses through the game. Bailey, Wise & Bolls (2009) showed that there is a strong link between self-designed avatars and game enjoyment, as players identify with and become invested in their character. As an additional motivator, the Pirate Plunder login screen has players select their avatar from a list of all the avatars in their class.

2. Method

2.1 Participants

The participants were 91 children between 10 and 11 years old (M = 10.58, SD = 0.32) from a large primary school in northern England. They were largely inexperienced with Scratch (having had sporadic lessons throughout primary school) and had no experience with the custom blocks or cloning that were used as a measure of abstraction in this study.

2.2 Experimental design

The study followed a pre-to-post-test, partial-crossover, quasi-experimental design to measure for improvements using procedural abstraction in Scratch after playing Pirate Plunder (Figure 11). For the first phase of the study, the three groups were split into Pirate Plunder (intervention), spreadsheets (non-programming active control) and Scratch (programming active control). In phase 2, the two control groups then crossed over to Pirate Plunder and the intervention group to the spreadsheets curriculum. The crossover was done for ethical reasons (so that all participants had the opportunity to play the game) and for game analytics (with three times as many players playing the game, more concrete conclusions can be drawn from the data to address RQ2). It also enabled us to see whether the phase 1 control groups would experience the same improvement on the assessments after playing the game.

Participants were assessed for their Scratch baseline ability using a Scratch 'baseline' task, their ability to use procedural abstraction in Scratch through a different Scratch 'challenge' and a multiple-choice Scratch abstraction test, and their computational thinking ability using the Computational Thinking test (Román-González et al., 2016). After playing Pirate Plunder, all participants were given questionnaires to measure their confidence using Scratch and 45 of them took part in artifact-based interviews.

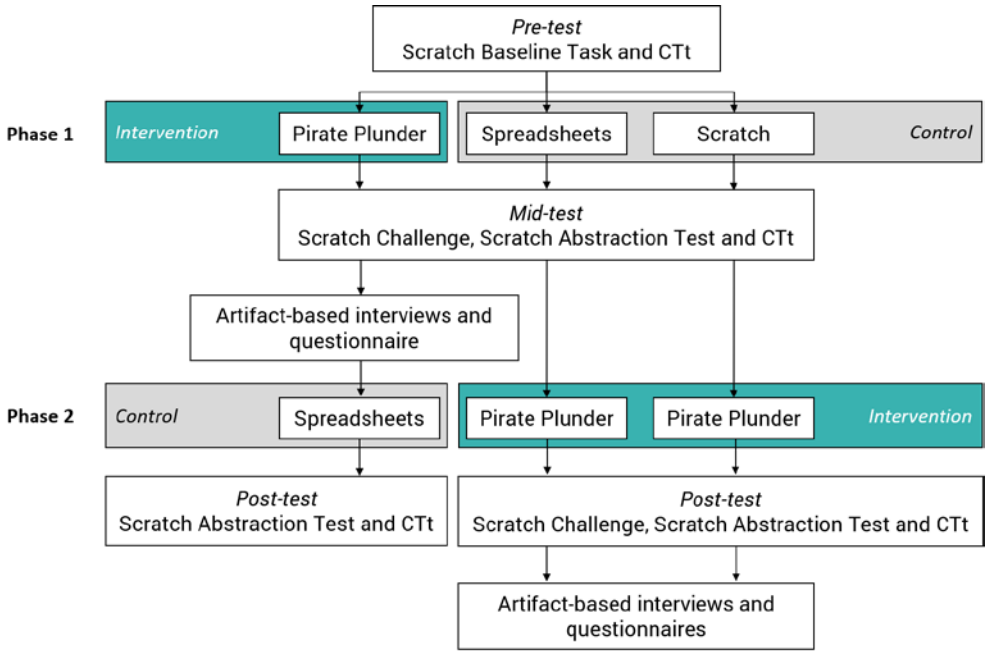

Figure 11: Diagram of the experimental design

2.3 Intervention materials

2.3.1 Pirate Plunder

Pirate Plunder was used for the intervention condition of the study. Participants played through the game at their own pace, with one researcher available to support those that had any difficulty. They continued each session where they left off in the previous session. Player performance is reported in Section 3.4.

2.3.2 Spreadsheets curriculum

The main comparison in the study was between Pirate Plunder and a spreadsheets curriculum designed for the age group produced by the UK-based educational resources company, Twinkl (2018). It was chosen as the primary control group activity because it did not involve programming (beyond using pre-made formula to calculate values, e.g. SUM and AVERAGE) or explicit computational thinking, yet still had participants using technology and being exposed to new learning content (Table 3).

Table 3: Spreadsheets curriculum lesson breakdown

Lesson Number	Lesson Name	Learning Content
1	Number Operations	Enter and edit text and numbers in cells and use SUM formula, begin formatting cells.
2	Ordering and Presenting Data	Using SUM formula for a specific purpose, ordering data using the sort function and producing graphs to present data.
3	Add, Edit and Calculate Data	Creating totals and averages on existing data, sorting and understanding the benefit of automatic recalculation.
4	Solving Problems	Investigating how to use a spreadsheet to solve a given problem.
5	Party Plan Budget	Choosing items for a party from a list of possible items and prices, using a spreadsheet to calculate quantities and totals within a set budget for a given number of people.
6	Design Your Own	Open-ended challenge to design their own spreadsheet.

2.3.3 Scratch curriculum

A Scratch curriculum (also produced by Twinkl) was used as a secondary comparison during phase 1. It involved creating an animated story based on a 'haunted house' and did not cover custom blocks or cloning (Table 4).

Table 4: Scratch animated stories curriculum lesson breakdown

Lesson Number	Lesson Name	Learning Content	Scratch/Programming Concept
1	Animate a Scene	Animating characters to around a scene	Green flag events, sounds, repeats (loops), changing size, gliding to position
2	Broadcast a Message	Using message broadcasting (sending and receiving messages) to sequence events	Message broadcasting
3	Show and Hide	Using show and hide to set the visibility of sprites	Show and hide
4	Sequence a Story	Creating a story (using a storyboard) with different backdrops	Backdrops, speech
5	Adding Audio	Recording and adding audio to the project	Sounds
6	Getting interactive	Using key press events to add extra functionality	Key press events

Both curricula were delivered by the first author and contained six-hours of learning content, the same amount of time that participants were given to play Pirate Plunder during the intervention. The possibility of potential experimenter bias is discussed in Section 4.5.1.

These two control groups were used to represent usual practice for computing lessons in England, representing a tougher test than usual (non-computing) lessons. They were not taught procedural abstraction as it is not part of the

curriculum for this age group, with the aim of the study being whether they could learn it through a game-based approach. Comparing Pirate Plunder with a procedural abstraction curriculum delivered using traditional teaching is the aim of a future study (Section 5.1).

2.4 Instruments and measures

2.4.1 Scratch baseline task

The Scratch baseline task was designed by the first author to allow participants to demonstrate Scratch proficiency, but to a specification that involved duplication, enabling them to use procedural abstraction if they were able to. The task involved animating the cat sprite around the edges of a rectangle, leaving an object on each corner. Figure 12 shows the project that participants started with. The Scratch baseline task was used at pre-test instead of the Scratch challenge assessment because it enables participants to achieve an outcome without prior Scratch knowledge. Whereas the Scratch challenge (Section 2.4.2) requires specific functionality that the participants had not been taught before the study.

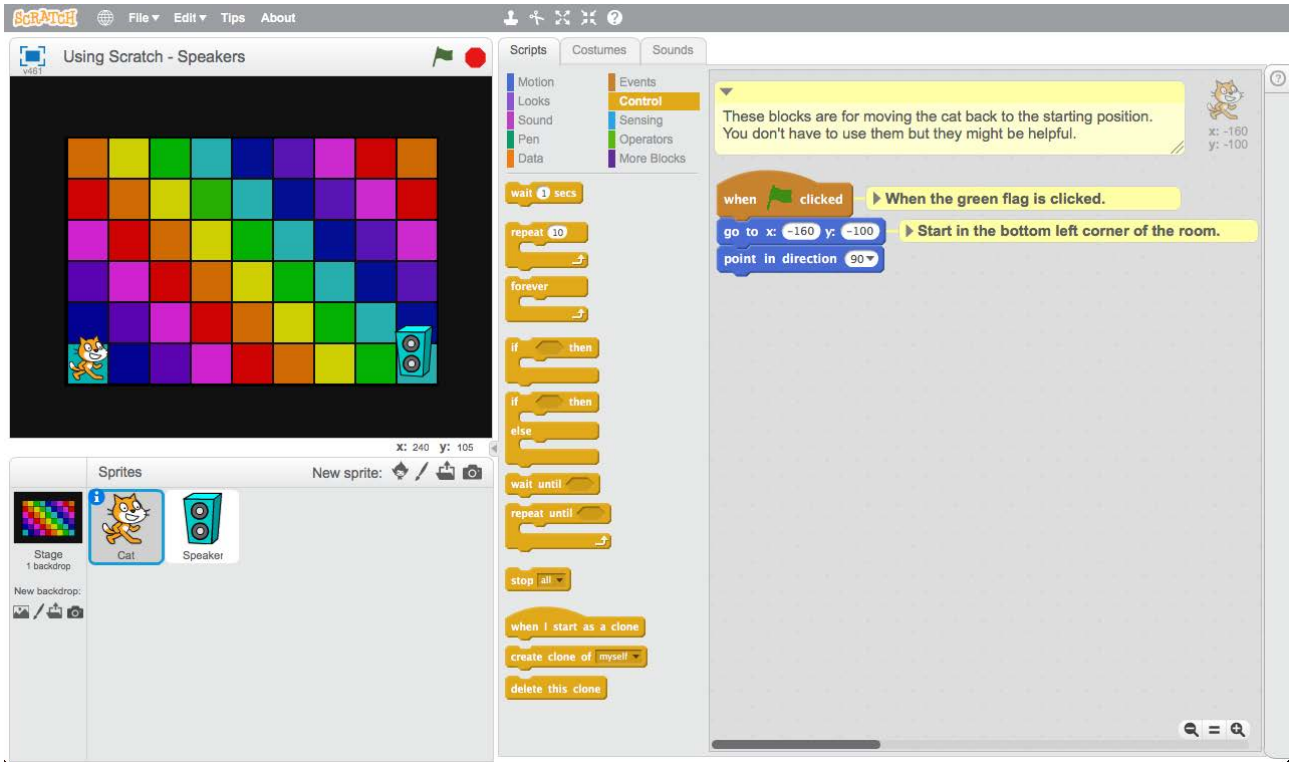

Figure 12: Scratch baseline task starter project

Both the Scratch baseline task and Scratch challenge were assessed using the Dr. Scratch automated assessment tool (Moreno-León & Robles, 2015), as it gives a quantitative score for 'abstraction and decomposition' (Table 5).

Table 5: Dr. Scratch abstraction and decomposition scoring system

Score	Required Functionality
1	More than one script and more than one sprite
2	Custom blocks
3	Cloning

2.4.2 Scratch challenge

The Scratch challenge was designed by the authors to see if participants could use procedural abstraction to reduce the block count in a pre-made project that contained both duplicated blocks and sprites (Figure 13). The project animates a cat sprite around a map, leaving a lamp post sprite on each corner. The optimal solution uses custom blocks and cloning to reduce the number of blocks and sprites but still achieve the same functionality (Figure 14). As with the Scratch baseline task, the Scratch challenge was assessed using the Dr. Scratch abstraction and decomposition score (Table 5).

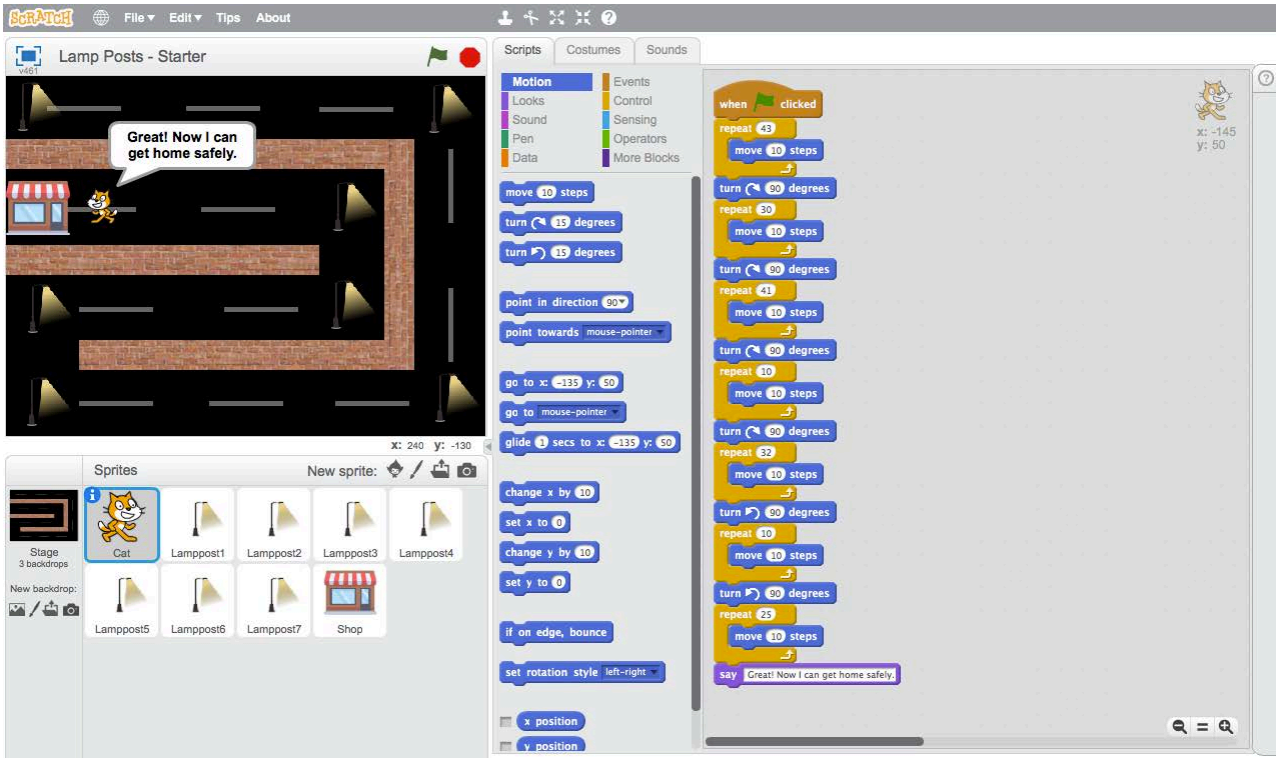

Figure 13: Scratch challenge starter project

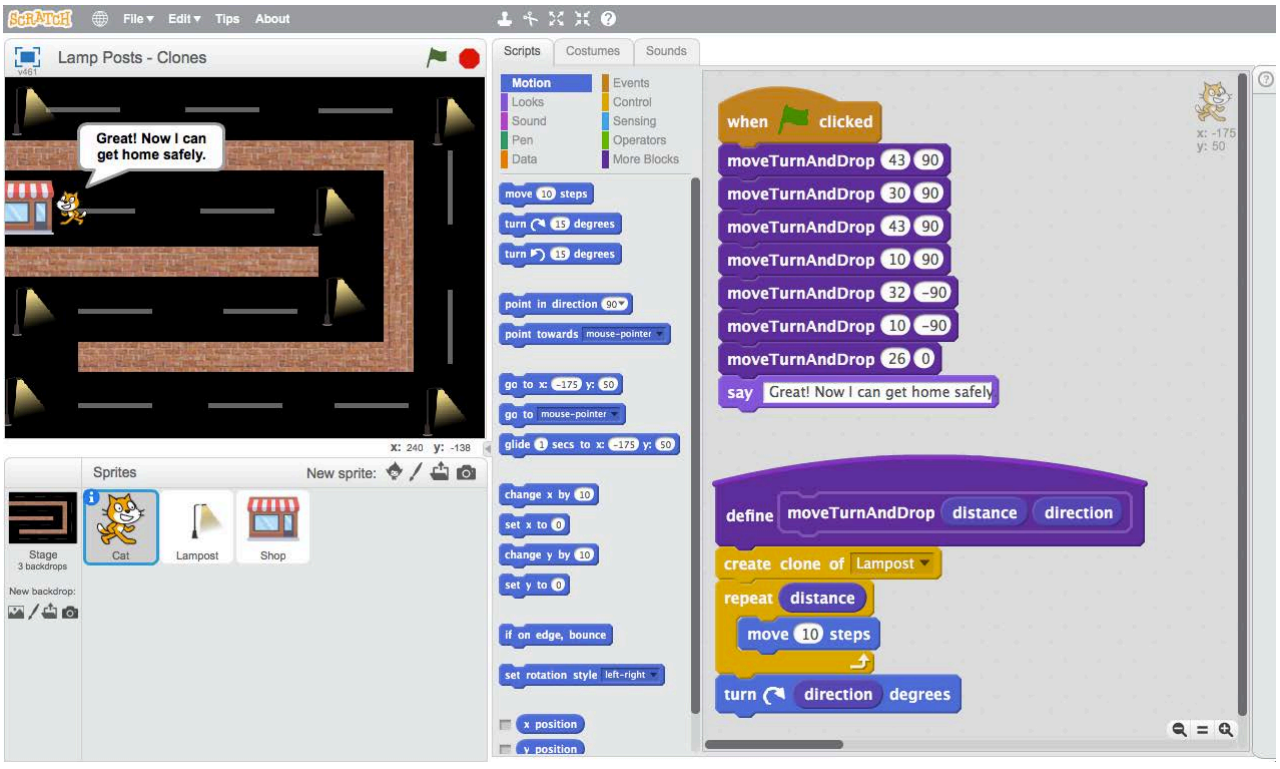

Figure 14: Scratch challenge optimal solution

Projects were also manually analysed against 'completeness criteria' (an explanation of each is given below) as a measure of whether the project had been completed using an optimal solution. This is because Dr. Scratch can only measure whether a block has been used in a project, not whether it has been used properly.

Completeness criteria:

1. Correct custom block - A custom block with two inputs representing distance and degrees (may not be named correctly), containing a 'repeat' 'move' for the distance and a 'turn' for the direction.
2. Correct use of cloning - A single lamp post sprite that is cloned at the position of the cat sprite inside the custom block before the repeat (because the starter project has a lamp post sprite at the starting position.)

3. Complete movement - The cat sprite is animated around the map and reaches the shop as it does in the starter project.
4. Correct lamp post positions - All the lamp post sprites are in the same positions as they are in the starter project. They must appear in sequence (ideally as the cat sprite reaches them.)

2.4.3 Multiple-choice Scratch abstraction test

The Scratch abstraction test was a 10-question multiple-choice assessment designed by the first author and used to supplement the Scratch challenge. The questions are on using custom blocks and cloning correctly in Scratch. Each question has four options with one correct answer. The test included questions on:

- Identifying duplicated Scratch code that can be refactored using a custom block.
- Identifying correct block names and inputs for duplicated code (Figure 15 shows a sample question).
- Comparing Scratch scenes and figuring out which sprites should be cloned.
- Identifying the block that can be used to get properties of a sprite.
- Identifying the blocks used to clone sprites successfully.

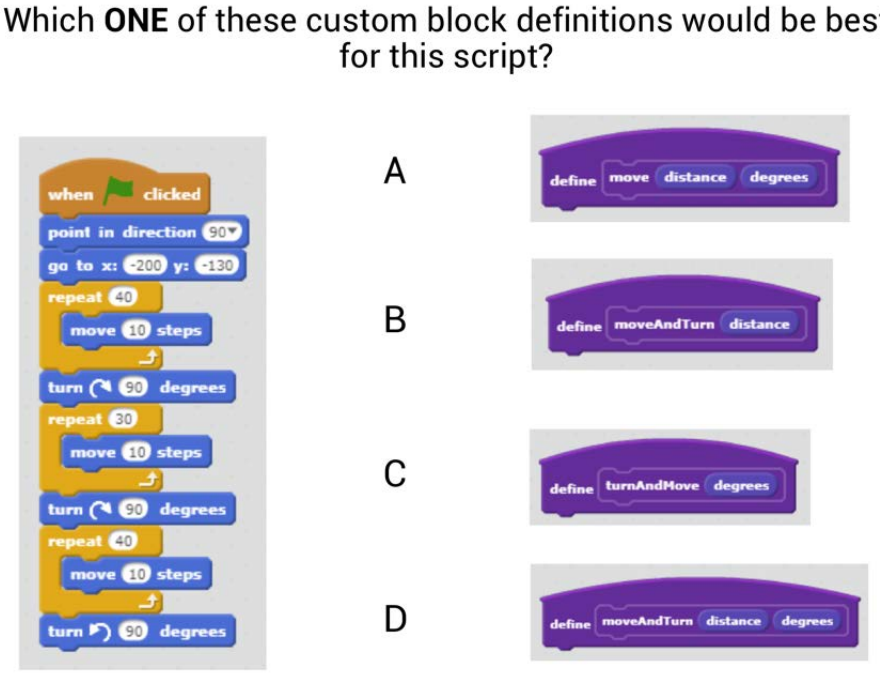

Figure 15: Sample question from the multiple-choice Scratch abstraction test (the correct answer is D)

2.4.4 Computational Thinking test

The Computational Thinking test (CTt) (Román-González et al., 2016) was used as a measure of computational thinking. It aims to measure "the ability to formulate and solve problems by relying on the fundamental concepts of computing, and using logic-syntax of programming languages: basic sequences, loops, iteration, conditionals, functions and variables" (p. 4). The CTt contains 28 multiple choice questions that use visual arrows or blocks common in educational programming tools.

Ideally, the CTt would be combined with another computational thinking assessment that does not use programming syntax, such as Bebras (Dagiene & Stupuriene, 2016). However, this was not possible in this study due to school logistics and time constraints.

2.4.5 Artifact-based interviews

Artifact-based interviews (Brennan & Resnick, 2012) were used to establish whether participants had understood the rationale for using procedural abstraction in the Scratch challenge. The interviews were one-to-one with a researcher and took place the day after the assessment. They began with open questions about the participant's project, to see if they could explain why they had done something without prompting from the researcher, before progressing to more leading questions about custom blocks and cloning. They were asked about their project (what each of the blocks did and why they had used them), alternative approaches they considered, why they had/had not used custom blocks,

why they had/had not used cloning, alternative scenarios in which they would use custom blocks or cloning, before finishing on similarities between the task and Pirate Plunder and general feedback on the game.

To select participants for the interviews, each intervention group was divided into three categories: correct solution, almost correct or interesting solution and no use of procedural abstraction. Five participants were selected from each category.

2.4.6 Game analytics

Pirate Plunder produces analytics for player actions. Table 6 shows when and why this data is produced, with the overall aim of figuring out why the game is effective (RQ2) (Section 4.3).

Table 6: Pirate Plunder analytics and their purpose

Analytic	Information	Purpose
Game section change (e.g. level select, shop, level)	Old section, new section and time spent on the section	Calculate how much time was spent on each section, mainly how much time was spent playing the game itself.
Level attempt (program execution)	Current program state, fast forward on/off, block count, program errors and time spent on the attempt	Establish common difficulties on levels.
Level completion	Time spent on the level, stars collected (score), attempts, block count and hints used	Establish player success on each level (based on score and block count), how useful the hints were and how difficult the levels were.
Program manipulation (block creation, move or deletion)	Manipulated block, old position and new position	Calculate how much program manipulation players were performing before coming to a solution that they would execute.
Purchasing shop items	Item purchased and cost	Establish when and how much players were spending on the shop.

2.5 Procedure

All participants completed the Scratch baseline task and the CTt at pre-test. The Scratch baseline task took place in the school IT suite in class groups. Participants were introduced to the study and the assessment task. They were then given 40 minutes to produce a Scratch project to the assessment specification. The CTt was administered using tablets in a classroom after the group had completed the Scratch baseline task. Participants were given a maximum of 45 minutes to complete the test.

Class groups were then assigned to the intervention (Pirate Plunder) or active control conditions (spreadsheets and Scratch) for phase 1 of the study. Both phases were four weeks long with two sessions per week (30 minutes and 50 minutes), taking place in the school IT suite.

At mid-test, all participants did the Scratch challenge (as opposed to the Scratch baseline task), multiple-choice Scratch abstraction test and the CTt. The intervention group also completed a questionnaire and 15 of them were interviewed. Once again, the Scratch challenge took place in the IT suite with participants given 40 minutes to modify the starter project. Both the multiple-choice Scratch abstraction test and CTt (in that order) were then administered using tablets in the classroom. They were given a maximum of 15 minutes for the multiple-choice abstraction test and 45 minutes for the CTt.

The conditions were then crossed over so that the intervention group did spreadsheets and the two control groups from phase 1 did Pirate Plunder. At post-test, the intervention groups re-completed the Scratch challenge, multiple-choice Scratch abstraction test and the CTt, whilst the control group only did the multiple-choice test and the CTt (Figure 11). Thirty of the phase 2 intervention participants were then interviewed across the following two days.

2.6 Data analysis

2.6.1 Hypotheses

The study had two hypotheses, both tested using the data from phase 1:

1. Pirate Plunder would perform better on the procedural abstraction measures in comparison with non-programming (spreadsheets) and programming (Scratch curriculum).
2. Pirate Plunder would improve scores on the CTt in comparison with the non-programming control group who were not doing explicit computational thinking activities.

The first hypothesis is tested using the Scratch challenge and multiple-choice Scratch abstraction test, using the Scratch baseline task as a covariate on the Scratch challenge between-groups comparison. The second hypothesis is tested using a between-groups comparison between the CTt scores from pre-to mid-test.

2.6.2 Statistical methods

To test the first hypothesis, we use a one-way ANCOVA to compare the mid-test Scratch challenge Dr. Scratch abstraction and decomposition scores for each group using the Scratch baseline task Dr. Scratch abstraction and decomposition scores as a covariate, to control for variance in baseline ability. In addition to a one-way ANOVA to compare the multiple-choice Scratch abstraction test mid-test scores.

For the second hypothesis, we use a one-way ANOVA to measure for a between-groups difference, then a series of independent samples t-tests to test for significant pairwise comparisons.

2.6.3 Research questions

RQ1 is addressed using the procedural abstraction measures (Scratch challenge and multiple-choice Scratch abstraction test) and is directly linked to the first hypothesis. RQ2 is addressed using the Pirate Plunder game analytics.

3. Results

3.1 Phase 1

3.1.1 Scratch challenge

Figure 16 shows the mean Dr. Scratch abstraction and decomposition scores on the Scratch challenge for each group at mid-test (note that the starting project gets 1 point for abstraction in Dr. Scratch). As stated in Section 2.6.2, a one-way ANCOVA was performed on the Scratch challenge abstraction and decomposition scores between the three groups, using the Scratch baseline task scores at pre-test as a covariate. The assumption of homogeneity of regression slopes was satisfactory (p = .89). The ANCOVA showed a significant difference in abstraction scores between the three groups, $F_{2, 78}$ = 30.30, p < .001, η^2 = .44. Table 7 shows the descriptive statistics.

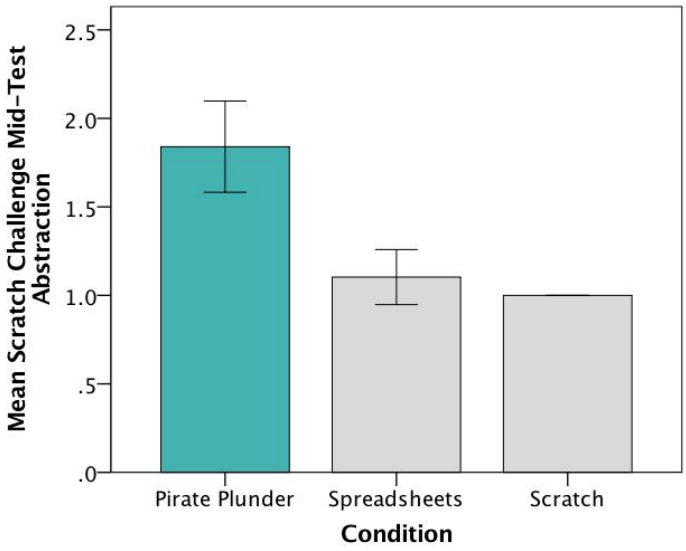

Figure 16: Comparison of the Dr. Scratch abstraction and decomposition scores on the mid-test Scratch challenge for each group (error bars show 95% confidence interval)

Table 7: Descriptive statistics of the Dr. Scratch abstraction and decomposition scores on the Scratch task and Scratch challenge for each group on phase 1

Condition		Scratch baseline task (pre-test)	Scratch challenge (mid-test)
Pirate Plunder	*M*	1.08	1.84
	N	25	25
	SD	0.70	0.62
Spreadsheets	*M*	0.93	1.10
	N	29	29
	SD	0.37	0.41
Scratch	*M*	1.11	1.00
	N	28	28
	SD	0.40	0.00

3.1.2 Multiple-choice Scratch abstraction test

There was a significant difference in the multiple-choice Scratch abstraction test scores between the three groups using a one-way ANOVA ($F_{2, 80} = 11.64$, $p < .001$, $\eta^2 = .23$), with the Pirate Plunder group ($M = 5.21$, $N = 28$, $SD = 1.40$) scoring significantly higher than both the non-programming ($M = 3.58$, $N = 26$, $SD = 1.86$) and programming control ($M = 3.45$, $N = 29$, $SD = 1.30$). Figure 17 shows the mean scores for each group (maximum possible score of 10).

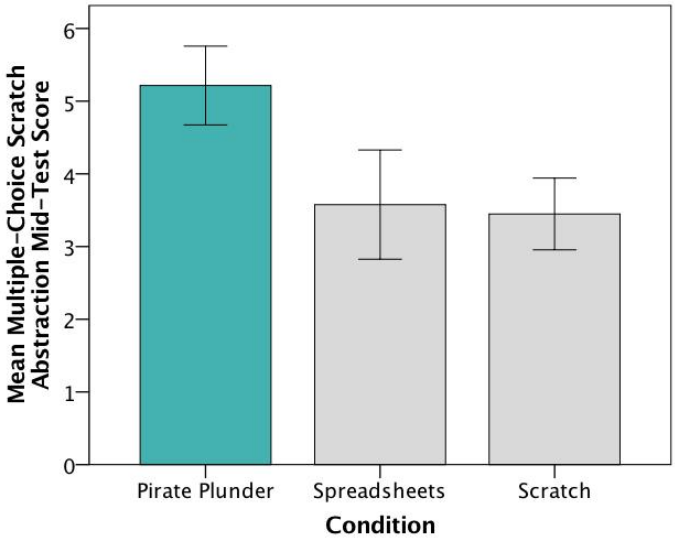

Figure 17: Comparison of the mid-test multiple-choice Scratch abstraction test scores for each group (error bars show 95% confidence interval)

3.1.3 Computational Thinking test

Figure 18 shows the mean Computational Thinking test learning gains from pre-to mid-test for the three groups. There was a significant difference between them ($F_{2, 84} = 3.72$, $p = .028$, $\eta^2 = .081$), with the only significant pairwise-comparison between the intervention group and the non-programming control: $t_{55} = 2.87$, $p = .015$, $d = 0.67$. Table 8 shows the descriptive statistics.

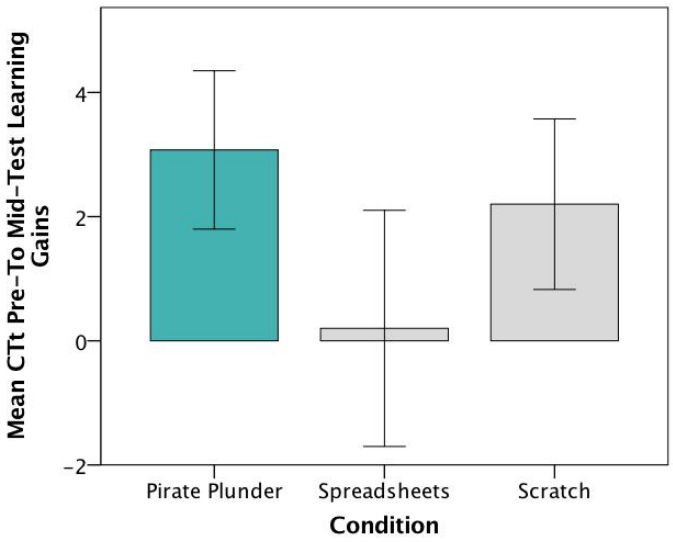

Figure 18: Comparison of the CTt learning gains from pre-to mid-test for each group (error bars show 95% confidence interval)

Table 8: Descriptive statistics of the Computational Thinking test from pre-to mid-test for each group (maximum score of 28)

Condition		Pre-test	Mid-test	Learning gains
Pirate Plunder	*M*	14.26	17.33	3.07
	N	27	27	27
	SD	5.9	5.61	3.22
Spreadsheets	*M*	14.33	14.53	0.20
	N	30	30	30
	SD	5.00	5.44	5.1
	M	15.70	17.90	2.20
	N	30	30	30
Scratch	*SD*	4.33	3.94	3.68

3.2 Phase 2

In phase 2 we would expect to see an improvement on the procedural abstraction measures for the groups who played Pirate Plunder. Due to the crossover design, there was not a genuine control group for phase 2 (as the control group had already been exposed to the intervention at this stage). As such, this section uses within-group comparative measures from before and after Pirate Plunder, instead of comparisons between groups. The groups are identified using their phase 1/phase 2 learning content (e.g. Pirate Plunder/spreadsheets).

3.2.1 Scratch challenge

Both phase 2 intervention groups improved significantly from mid-to post-test: spreadsheets/Pirate Plunder (t_{28} = 5.52, p < .001, d = 1.44) and Scratch/Pirate Plunder (t_{25} = 8.76, p < .001, d = 1.44). The phase 2 control group did not re-complete the assessment at post-test due to school limitations.

3.2.2 Multiple-choice Scratch abstraction test

A one-way ANOVA showed that there was no significant difference between groups: $F_{2, 71}$ = 2.21, p = .12, η^2 = .059. In addition, using a paired samples t-test, only the Scratch/Pirate Plunder group changed (in this case, improved) significantly from mid-to post-test: t_{26} = 2.14, p = .042, d = 0.47.

3.2.3 Computational Thinking test

A one-way ANOVA showed a significant difference between groups, $F_{2, 84}$ = 4.49, p = .014, η^2 = .097. A paired samples t-test showed that the Pirate Plunder/spreadsheets group declined significantly: t_{27} = 2.87, p = .008, d = 0.38.

3.3 Scratch challenge completeness criteria after playing Pirate Plunder

The Scratch challenge completeness criteria (Section 2.4.2) were used to support the Dr. Scratch abstraction and decomposition scores. Table 9 shows the number of participants that met each criterion in their Scratch challenge projects before and after their Pirate Plunder intervention. There were significant improvements for the phase 2 intervention groups from mid-to post-test in using the correct custom block (spreadsheets/Pirate Plunder, $t_{28} = 6.84$, $p < .001$, $d = 1.80$ and Scratch/Pirate Plunder, $t_{25} = 9.21$, $p < .001$, $d = 2.56$) and the correct use of cloning (spreadsheets/Pirate Plunder, $t_{28} = 2.42$, $p = .023$, $d = 0.63$ and Scratch/Pirate Plunder, $t_{25} = 2.13$, $p = .043$, $d = 0.58$).

Table 9: Scratch challenge completeness criteria before and after Pirate Plunder intervention

Condition	*N*		Correct custom block		Correct use of cloning		Complete movement		Correct lamp post positions	
	Before	After	Before	After	Before	After	Before	After	Before	After
Pirate Plunder/Spreadsheets	-	25	-	10	-	0	-	16	-	21
Spreadsheets/Pirate Plunder	29	29	0	11	0	5	9	19	22	26
Scratch/Pirate Plunder	26	26	0	13	0	4	8	22	9	15

3.4 Pirate Plunder player performance

Table 10 shows the Pirate Plunder player performance for each group. This was judged using the number of challenges completed (maximum of 40) and overall stars collected (maximum of 120). The average stars collected on each level (maximum of 3) is also given. One-way ANOVAs showed no significant difference between the three groups for challenges completed ($F_{2, 87} = 0.81$, $p = .447$, $\eta^2 = .018$) or stars collected ($F_{2, 87} = 1.13$, $p = .329$, $\eta^2 = .025$).

Table 10: Descriptive statistics for Pirate Plunder player performance

Condition		Challenges completed	Total stars collected	Average stars per level
Pirate Plunder/Spreadsheets	*M*	33.00	95.72	2.89
	N	29	29	29
	SD	6.51	21.84	0.21
Spreadsheets/Pirate Plunder	*M*	32.23	94.90	2.94
	N	30	30	30
	SD	5.93	18.73	0.1
Scratch/Pirate Plunder	*M*	34.13	101.52	2.97
	N	31	31	31
	SD	5.05	15.40	0.05

3.5 Artifact-based interview observations

When asked if they could give another example in which they would use a custom block in Scratch, most participants (29/45, 64.4%) gave examples situated in the context that they had learnt to use procedural abstraction (i.e. involving moving and turning a sprite) (category A). However, 12 participants (26.7%) were able to apply procedural abstraction to theoretical scenarios outside of Pirate Plunder or could explain general rules when using procedural abstraction (category B). For example, one participant said that they could use custom blocks and cloning when creating a bowling game. Even going as far as to question whether cloning would be appropriate or not due to the way it works:

> *Researcher:* "Can you give me another example of where you'd use a custom block in Scratch?"
> *Participant:* "You could create a bowling game and you could input the amount of power the ball would move, so you could determine how far it would go, or you could use it for some sort of game where you'd throw or catapult something. So, you could change at different moments how far it would go."

The remaining four participants (8.9%) struggled to apply procedural abstraction to any scenario (even if they had used it in the Scratch challenge and/or the game) (category C). These categories (when scored 1-3) correlate significantly with participant CTt pre-test scores, $r_{45} = .34$, $p = .021$. Table 11 shows the descriptive statistics.

Table 11: Average CTt pre-test score for interview observation category

Category (score)	CTt pre-test mean	*N*	*SD*

A (2)	15.72	29	5.14
B (3)	19.25	12	4.25
C (1)	13.75	4	4.57

4. Discussion

4.1 RQ1 - Can a game-based learning approach be used to teach primary school children to use procedural abstraction in Scratch projects?

The phase 1 results support the first hypothesis: children's improvements on measures of procedural abstraction after playing Pirate Plunder were greater in comparison with the control groups. In phase 2, the children that played Pirate Plunder significantly improved their scores on the Scratch challenge.

In addition, the Scratch challenge completeness criteria show that the game was effective in getting children to use procedural abstraction correctly. But, less so than the Dr. Scratch abstraction and decomposition scores indicate. Sixty-one participants achieved a Dr. Scratch abstraction and decomposition score of 2 or 3 (for using custom blocks or cloning) on the Scratch challenge after playing Pirate Plunder. Yet, only 34 of these produced the correct custom block and nine produced the correct cloning solution. This is a weakness of using Dr. Scratch as a measure in a study such as this because it does not account for whether a block or functionality has been used correctly.

This addresses RQ1 by showing that primary school children (age 10 and 11) can be taught to use procedural abstraction in Scratch projects using a game-based learning approach.

4.2 Computational thinking

The results of the Computational Thinking test support hypothesis 2, that Pirate Plunder improved participants' computational thinking compared to the non-programming control after phase 1 of the study. Yet, these results were not repeated after the crossover. The decline in the Pirate Plunder/spreadsheets group in phase 2 is likely because the participants were doing the same assessment for the third time and had lost some motivation to complete it properly.

There are, however, issues with using programming-based assessments as measures of computational thinking, as they do not consider the wider use of computational thinking in problem-solving (Kazimoglu et al., 2011).

4.3 RQ2 - What aspects of the game design influence the effectiveness of this approach and why?

In this section, we give four reasons why Pirate Plunder was effective that are supported by the game analytics (Section 2.4.6).

4.3.1 Motivating players using restrictive success conditions

For players to experience flow (Csikszentmihalyi, 1990), games must maintain the balance between the challenge of the levels and ability of the player, whilst also providing clear goals and immediate and accurate feedback. Pirate Plunder was effective in keeping players motivated using these flow enablers. On average, players completed 82.8% (33.12/40) of the challenge levels by the end of the intervention. Additionally, they spent 80.07% of their time playing through the levels (as opposed to time spent on the shop, class screen or level select).

As described in Section 1.4.2, restrictive success conditions such as block limits (the number of blocks a player can use to complete a level) and required block validation (completing the level using the block linked to that challenge) were used to force the player into producing optimal solutions. This was an effective approach, despite observations that players often tried to circumvent program restrictions. Players had an average score of 2.93/3 stars per challenge level, meaning that they were using the correct blocks and producing optimal solutions (in terms of block count) 97.66% of the time. In addition, players tended not to reattempt levels once they had completed them (the average number of total challenge reattempts per player was 3.1), meaning that the high average scores were achieved the first time the player completed the level.

4.3.2 Effective customisation system

Customisation is a powerful motivator in both learning (Cordova & Lepper, 1996) and games (Turkay & Adinolf, 2015). In Pirate Plunder, it is used together with a reward system, where players collect coins for completing levels successfully (Section 1.4.4). The game analytics showed that this was an effective design strategy. Players spent 79.36% of their total coin earnings on purchasing items and customising their avatars. Players also purchased items

regularly throughout gameplay. Across the eight sessions that each player was given, players purchased an average of 3.14 items and spent 84.33 coins per session (Figure 19). This shows that the coin rewards for playing the game were an effective method of motivating players to continue through the learning content, as was the unlocking of items as the player progressed through the challenge levels. This was the case even though avatar items do not give the player an advantage in the game and are for aesthetics only.

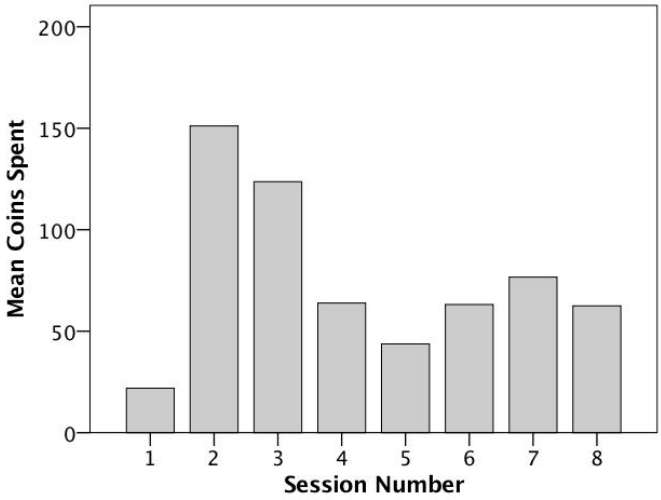

Figure 19: Bar chart showing the average number of coins spent in each session

4.3.3 Allowance for different levels of prior knowledge

Papert (1980) stated that educational programming tools should have a low floor (easy access) and a high ceiling (vast potential) to be inclusive for all learners. The Pirate Plunder learning trajectory (Section 1.4.2) follows these principles, introducing basic Scratch blocks and functionality first. This was an effective approach in lowering the barrier of entry for the game. Players who achieved less than the mean score (14.85) on the CTt at pre-test still completed 74.9% (29.97/40) of the challenges (7.9% lower than the mean). Additionally, the game also had a suitably high ceiling. None of the 20 participants who completed all 40 levels introducing the learning content were able to complete all eight of the 'general levels' (Section 1.4.2) that become available after level 40 is complete. The CTt pre-test is used here because it is a more general measure of programming ability than the Scratch-based assessments.

4.3.4 Effective difficulty curve

Difficulty scaling is a fundamental part of game design (Aponte et al., 2009). Despite not being adjusted dynamically on a per-player basis, the Pirate Plunder difficulty progression fits with tension-resolution cycles linked to player enjoyment: when a new concept is introduced, the player initially feels tension until they gain an understanding of the concept (resolution). Figure 20 shows that the difficulty curve was effective in introducing new concepts without them being too difficult, with players completing introductory levels quickly using few attempts. There is a jump for the last 'loops' challenge (level 15) (Figure 5), then a larger jump when procedures are introduced (level 19). The dip after this introduction shows that the 'custom block' and 'inputs' levels introduced further complexity successfully without being too difficult.

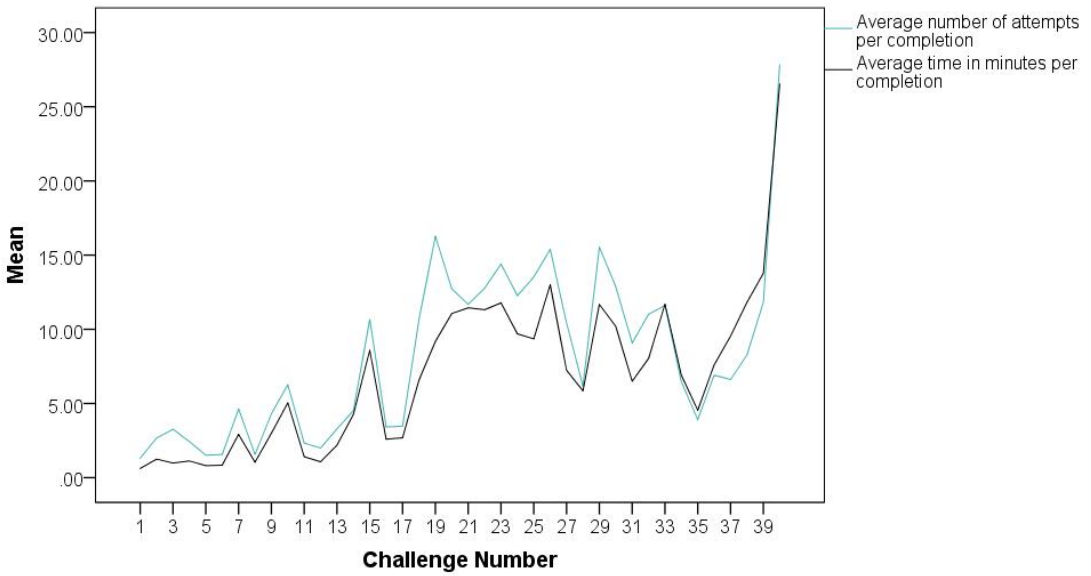

Figure 20: Line graph of the average attempts per challenge and the average time in minutes per challenge (both per level completion)

4.4 Recommendations

In this section, we draw out three recommendations for educational game design from our research.

4.4.1 Using learning trajectories and restrictive success conditions to introduce complex learning content

Educational games allow for granular management of when and how learning content is introduced. Structuring this effectively is an important part of educational game design. The results of this study and the reasons the game was effective (Section 4.3) show that learning trajectories, such as the one we have used to introduce procedural abstraction, are an effective method of introducing conceptually difficult learning content. By designing levels around this, the learner can use the concept without instruction from the teacher, because the learning content is introduced and reinforced by the game. Restrictive success conditions on levels, such as block limits and validation conditions, are important in getting players to use taught functionality correctly. This is particularly important in primary education, where teachers often lack theoretical and technical knowledge of computing.

4.4.2 Increasing learner investment through customisable avatars

Motivating learners to progress through the content is vital in educational games. Particularly when the novel learning content is introduced in the latter parts. The regularity and amount of player spending (Section 4.3.2), combined with the high percentage of maximum scores on Pirate Plunder levels (Section 4.3.1), shows that players were invested in their avatars and this served as a good motivator for success. We, therefore, recommend that avatars, where the player can upgrade them using currency or rewards earnt playing the game, are a good method of increasing learner investment. This combines the extrinsic motivator of purchasing items for the avatar with the intrinsic motivator of collecting the maximum number of coins possible by completing levels.

In earlier versions of Pirate Plunder, we placed a small cost on saving changes to an avatar to ensure that players did not spend too much time customising their avatar early in the game (when they have a limited number of coins). However, these fears were unfounded, and we removed this for the study reported in this article.

4.4.3 Improving evaluations of educational games

One of the strengths of this work is that Pirate Plunder was evaluated against existing curricula that are used in English primary schools, including both programming (Scratch) and non-programming (spreadsheets) lesson plans, using a range of quantitative and qualitative assessments.

Two large literature reviews of evaluations of educational games in the classroom (Hainey et al., 2016; Petri & Gresse von Wangenheim, 2017) state that whilst there are many examples of strong study designs, either randomised controlled trials or quasi-experimental studies, there are a larger number of studies that use simple, ad-hoc research designs, subjective feedback via questionnaires and small sample sizes to evaluate educational games.

The combination of the range of educational programming tools available (Section 1.1.1) and the increasing pressure to deliver computer science learning content, mean that robust evaluations of programming games are essential. Otherwise, it is difficult to know whether these tools are effective in a real-world classroom environment.

4.5 Limitations

4.5.1 Potential experimenter bias

As the lead author delivered both the intervention and control group content, there is a possibility that experimenter bias could have played a role in the outcomes of the study. Ideally, the control group content would have been delivered by a teacher blind to the hypothesis. We aim to address this issue in future empirical work (Section 5.1).

4.5.2 Situated learning

The participants used procedural abstraction in a situated context, namely 'moving' and 'turning' in Scratch, with the majority then only able to explain procedural abstraction within that context (Section 3.5). However, the finding that they can learn to use procedural abstraction in any context is important and one that to our knowledge has not been demonstrated by primary school children before (RQ1). In addition, there were instances of higher scoring participants (on the CTt at pre-test) being able to explain how they could apply procedural abstraction in other Scratch projects after playing Pirate Plunder. We aim to address this issue in future work by having participants use Scratch to produce projects using procedural abstraction after playing Pirate Plunder.

5. Conclusions

In conclusion, we have addressed RQ1 in showing that Pirate Plunder can be used to teach primary school children (age 10 and 11) to use procedural abstraction in Scratch projects. This is a significant finding and indicates that procedural abstraction can be part of computer science curricula for this age group, supporting the results of Kalas & Benton (2017).

Using the game analytics, we have then addressed RQ2 by giving reasons why the game was effective. Firstly, because players were motivated to progress through the content using our learning trajectory and restrictive success conditions. Secondly, customisation was effective in motivating players to continue playing the game. Thirdly, the game allows for players with less prior knowledge to still progress through the game. Finally, the difficulty curve is effective in introducing new concepts. This then led into three recommendations for designing programming games to support computer science knowledge: using learning trajectories and restrictive success conditions to introduce complex learning content, increasing learner investment through customisable avatars and improving the evaluation of educational games.

The success of Pirate Plunder shows that game-based learning can play a key role in supporting and delivering computer science content in primary education. We hope that our recommendations will be used by game designers to improve the delivery of learning content in educational games.

5.1 Future work

Future empirical work will address the limitations of this study. Firstly, to conduct studies in multiple schools to confirm our findings with a more generalisable sample. Secondly, to have the control group content delivered by a teacher or researcher blind to the hypothesis, to remove any potential experimenter bias. We also aim to evaluate Pirate Plunder against a traditional computer science curriculum that introduces procedural abstraction, as opposed to standard computing curricula. This will allow us to evaluate whether the game is effective in comparison with another form of instruction. Further to this, we would like to explore whether the procedural abstraction skills learnt in Scratch (using Pirate Plunder) transfer to other Scratch projects, as indicated by the artifact-based interviews and then onto text-based languages, whilst investigating how this transfer can be effectively mediated.

In terms of development, we aim to extend Pirate Plunder to include other computer science concepts that novices struggle with, such as variables and conditionals (Grover & Basu, 2017). The game would also be updated for use alongside Scratch 3, which due to being a native web application, now allows for easier extension and project analysis (e.g. Stahlbauer et al., 2019).

Funding
This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

2Simple Ltd. (2020). *Purple Mash*. https://2simple.com/purple-mash/

Aivaloglou, E., & Hermans, F. (2016). How Kids Code and How We Know: An Exploratory Study on the Scratch Repository. *Proceedings of the 2016 ACM Conference on International Computing Education Research*, 53–61. https://doi.org/10.1145/2960310.2960325

Aponte, M.-V., Levieux, G., & Natkin, S. (2009). Scaling the Level of Difficulty in Single Player Video Games. *Proceedings of the 2009 International Conference on Entertainment Computing*, 24–35. https://doi.org/10.1007/978-3-642-04052-8_3

Bailey, R., Wise, K., & Bolls, P. (2009). How Avatar Customizability Affects Children's Arousal and Subjective Presence During Junk Food–Sponsored Online Video Games. *Cyberpsychology & Behavior*, *12*(3), 277–283. https://doi.org/10.1089/cpb.2008.0292

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable Programming: Blocks and Beyond. *Communications of the ACM*, *60*(6), 72–80. https://doi.org/10.1145/3015455

Bauer, A., Butler, E., & Popović, Z. (2017). Dragon Architect: Open Design Problems for Guided Learning in a Creative Computational Thinking Sandbox Game. *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 1–6. https://doi.org/10.1145/3102071.3102106

Brennan, K., & Resnick, M. (2012). New Frameworks for Studying and Assessing the Development of Computational Thinking. *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*, 1–25.

Clements, D. H., & Sarama, J. (2004). Learning Trajectories in Mathematics Education. *Mathematical Thinking and Learning*, *6*(2), 81–89. https://doi.org/10.1207/s15327833mtl0602_1

Code.org. (2020). *Code.org*. https://code.org/

Cooper, S., Dann, W., & Pausch, R. (2003). Teaching Objects-first in Introductory Computer Science. *ACM SIGCSE Bulletin*, *35*(1), 191. https://doi.org/10.1145/792548.611966

Cordova, D. I., & Lepper, M. R. (1996). Intrinsic Motivation and the Process of Learning: Beneficial Effects of Contextualization, Personalization, and Choice. *Journal of Educational Psychology*, *88*(4), 715–730. https://doi.org/10.1037/0022-0663.88.4.715

Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. Harper Perennial.

Dagiene, V., & Stupuriene, G. (2016). Bebras - a Sustainable Community Building Model for the Concept Based Learning of Informatics and Computational Thinking. *Informatics in Education*, *15*(1), 25–44. https://doi.org/10.15388/infedu.2016.02

Dasgupta, S., Hale, W., Monroy-Hernández, A., & Hill, B. M. (2016). Remixing as a Pathway to Computational Thinking. *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 1438–1449. https://doi.org/10.1145/2818048.2819984

Denning, P. J. (2017). Remaining Trouble Spots With Computational Thinking. *Communications of the ACM*, *60*(6), 33–39. https://doi.org/10.1145/2998438

Dorling, M., & White, D. (2015). Scratch: A Way to Logo and Python. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 191–196. https://doi.org/10.1145/2676723.2677256

Duncan, C., Bell, T., & Tanimoto, S. (2014). Should Your 8-Year-Old Learn Coding? *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 60–69. https://doi.org/10.1145/2670757.2670774

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison Wesley.

Google. (2020). *Google Blockly*. https://developers.google.com/blockly

Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 267–272. https://doi.org/10.1145/3017680.3017723

Grover, S., Jackiw, N., Lundh, P., & Basu, S. (2018). Combining Non-Programming Activities With Programming for Introducing Foundational Computing Concepts. *Proceedings of the International Society of the Learning Sciences*, 925–928. https://doi.org/10.22318/cscl2018.925

Haberman, B. (2004). High-School Students ' Attitudes Regarding. *Education and Information Technologies*, *9*(2), 131–145. https://doi.org/10.1023/B:EAIT.0000027926.99053.6f

Habgood, M. P. J., & Ainsworth, S. E. (2011). Motivating Children to Learn Effectively: Exploring the Value of Intrinsic Integration in Educational Games. *Journal of the Learning Sciences*, *20*(2), 169–206. https://doi.org/10.1080/10508406.2010.508029

Hainey, T., Connolly, T. M., Boyle, E. A., Wilson, A., & Razak, A. (2016). A Systematic Literature Review of Games-based Learning Empirical Evidence in Primary Education. *Computers and Education*, *102*, 202–223. https://doi.org/10.1016/j.compedu.2016.09.001

Heintz, F., Mannila, L., & Farnqvist, T. (2016). A Review of Models for Introducing Computational Thinking, Computer Science and Computing in K-12 Education. *Proceedings of the 2016 IEEE Frontiers in Education Conference*, 1–9. https://doi.org/10.1109/FIE.2016.7757410

Hermans, F., & Aivaloglou, E. (2016). Do Code Smells Hamper Novice Programming? A Controlled Experiment on Scratch Programs. *Proceedings of the IEEE 24th International Conference on Program Comprehension*, 1–10. https://doi.org/10.1109/icpc.2016.7503706

Hooshyar, D., Ahmad, R. B., Yousefi, M., Fathi, M., Horng, S. J., & Lim, H. (2016). Applying an Online Game-based Formative Assessment in a Flowchart-based Intelligent Tutoring System for Improving Problem-solving Skills. *Computers and Education*, *94*(November), 18–36. https://doi.org/10.1016/j.compedu.2015.10.013

Hooshyar, D., Lim, H., Pedaste, M., Yang, K., Fathi, M., & Yang, Y. (2019). AutoThinking: An Adaptive Computational Thinking Game. *ICITL 2019*, 381–391. https://doi.org/10.1007/978-3-030-35343-8_41

Hopscotch Technologies. (2020). *Hopscotch*. https://www.gethopscotch.com/

Kalas, I., & Benton, L. (2017). Defining Procedures in Early Computing Education. *Proceedings of the IFIP World Conference on Computers in Education*, *515*, 567–578. https://doi.org/10.1007/978-3-319-74310-3_57

Kallia, M., & Sentance, S. (2017). Computing Teachers' Perspectives on Threshold Concepts. *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, 15–24. https://doi.org/10.1145/3137065.3137085

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2011). Understanding Computational Thinking Before Programming: Developing Guidelines for the Design of Games to Learn Introductory Programming Through Game-play. *International Journal of Game-Based Learning*, *1*(3), 30–52. https://doi.org/10.4018/ijgbl.2011070103

Larke, L. R. (2019). Agentic Neglect: Teachers as Gatekeepers of England's National Computing Curriculum. *British Journal of Educational Technology*, *50*(3), 1137–1150. https://doi.org/10.1111/bjet.12744

Lazonder, A. W., & Harmsen, R. (2016). Meta-Analysis of Inquiry-Based Learning: Effects of Guidance. *Review of Educational Research*, *86*(3), 681–718. https://doi.org/10.3102/0034654315627366

Livingstone, I., & Hope, A. (2011). *Next Gen: Transforming the UK Into the World's Leading Talent Hub for the Video Games and Visual Effects Industries*.

Madison, S., & Gifford, J. (1997). Parameter Passing: The Conceptions Novices Construct. *Proceedings of the Annual Meeting of the American Educational Research Association*, 2–29.

Maloney, J., Resnick, M., & Rusk, N. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, *10*(4), 1–15. https://doi.org/10.1145/1868358.1868363

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of Programming in Scratch. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, 168–172. https://doi.org/10.1145/1999747.1999796

Meyer, J. H. F., & Land, R. (2003). Threshold Concepts and Troublesome Knowledge: Linkages to Ways of Thinking and Practising Within the Disciplines. In *Improving Student Learning – Ten Years On* (pp. 1–16).

Moreno-León, J., & Robles, G. (2014). Automatic Detection of Bad Programming Habits in Scratch: A Preliminary Study. *Proceedings of the 2014 IEEE Frontiers in Education Conference*, 1–4. https://doi.org/10.1109/FIE.2014.7044055

Moreno-León, J., & Robles, G. (2015). Dr. Scratch: a Web Tool to Automatically Evaluate Scratch Projects. *Proceedings of the Workshop in Primary and Secondary Computing Education*, 132–133. https://doi.org/10.1145/2818314.2818338

Neuron Fuel. (2020). *Tynker*. https://www.tynker.com/

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc.

Petri, G., & Gresse von Wangenheim, C. (2017). How Games for Computing Education Are Evaluated? A Systematic Literature Review. *Computers & Education*, *107*, 68–90. https://doi.org/10.1016/j.compedu.2017.01.004

Rich, P. J., Browning, S. F., Perkins, M., Shoop, T., Yoshikawa, E., Belikov, O. M., Rich, P. J., & Shoop, T. (2019). Coding in K-8: International Trends in Teaching Elementary/Primary Computing. *TechTrends*, *63*(3), 311–329. https://doi.org/10.1007/s11528-018-0295-4

Rijke, W. J., Bollen, L., Eysink, T. H. S., & Tolboom, J. L. J. (2018). Computational Thinking in Primary School: An Examination of Abstraction and Decomposition in Different Age Groups. *Informatics in Education*, *17*(1), 77–92. https://doi.org/10.15388/infedu.2018.05

Robles, G., Moreno-León, J., Aivaloglou, E., & Hermans, F. (2017). Software Clones in Scratch Projects: On the Presence of Copy-and-Paste in Computational Thinking Learning. *Proceedings of the 2017 IEEE 11th International Workshop on Software Clones*, 31–37. https://doi.org/10.1109/IWSC.2017.7880506

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2016). Which Cognitive Abilities Underlie Computational Thinking? Criterion Validity of the Computational Thinking Test. *Computers in Human Behavior*, *72*, 678–691. https://doi.org/10.1016/j.chb.2016.08.047

Román-González, M., Pérez-González, J.-C., Moreno-León, J., & Robles, G. (2018). Extending the Nomological Network of Computational Thinking With Non-cognitive Factors. *Computers in Human Behavior*, *80*, 441–459. https://doi.org/10.1016/j.chb.2017.09.030

Rose, S. P. (2019). *Developing Children's Computational Thinking using Programming Games*.

Rose, S. P., Habgood, M. P. J., & Jay, T. (2017). An Exploration of the Role of Visual Programming Tools in the Development of Young Children's Computational Thinking. *Electronic Journal of E-Learning*, *15*(4), 297–309.

Rose, S. P., Habgood, M. P. J., & Jay, T. (2019). Using Pirate Plunder to Develop Children's Abstraction Skills in Scratch. *Proceedings of the Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3290607.3312871

Rose, S. P., Habgood, M. P. J., & Jay, T. (2018). Pirate Plunder: Game-Based Computational Thinking Using Scratch Blocks. *Proceedings of the 12th European Conference for Game Based Learning*, 556–564.

Scratch Team. (2020). *Scratch Statistics*. https://scratch.mit.edu/statistics/

Seals, C., Rosson, M. B., Carroll, J. M., Lewis, T., & Colson, L. (2002). Fun Learning Stagecast Creator: An Exercise in Minimalism and Collaboration. *Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, 177–186. https://doi.org/10.1109/HCC.2002.1046370

Sherman, M., & Martin, F. (2015). The Assessment of Mobile Computational Thinking. *Journal of Computing Sciences in Colleges*, *30*(6), 53–59.

Stahlbauer, A., Kreis, M., & Fraser, G. (2019). Testing Scratch Programs Automatically. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*, 165–175. https://doi.org/10.1145/3338906.3338910

Statter, D., & Armoni, M. (2020). Teaching Abstraction in Computer Science to 7th Grade Students. *ACM Transactions on Computing Education*, *20*(1), 8–837. https://doi.org/10.1145/3372143

Techapalokul, P. (2017). Sniffing Through Millions of Blocks for Bad Smells. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 781–782. https://doi.org/10.1145/3017680.3022450

Techapalokul, P., & Tilevich, E. (2019a). Code Quality Improvement for All: Automated Refactoring for Scratch. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, *2019-Octob*, 117–125. https://doi.org/10.1109/VLHCC.2019.8818950

Techapalokul, P., & Tilevich, E. (2019b). Position: Reusing in the Small: Promoting Procedural Abstraction in Scratch Communal Learning. *Proceedings - 2019 IEEE Blocks and Beyond Workshop, B and B 2019*, 59–61. https://doi.org/10.1109/BB48857.2019.8941228

Techapalokul, P., & Tilevich, E. (2015). Programming Environments for Blocks Need First-Class Software Refactoring Support: A Position Paper. *Proceedings of the 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 109–111. https://doi.org/10.1109/BLOCKS.2015.7369015

The Royal Society. (2012). *Shut Down or Restart? The Way Forward for Computing in UK Schools*.

Turkay, S., & Adinolf, S. (2015). The Effects of Customization on Motivation in an Extended Study With a Massively Multiplayer Online Roleplaying Game. *Cyberpsychology*, *9*(3). https://doi.org/10.5817/CP2015-3-2

Twinkl Educational Publishing. (2018). *Twinkl*. https://twinkl.co.uk

Weintrop, D., & Wilensky, U. (2013). RoboBuilder: A Computational Thinking Game. *Sigcse*, 736. https://doi.org/10.1145/2445196.2445430

Werneburg, S., Manske, S., & Hoppe, H. U. (2016). ctGameStudio – A Game-Based Learning Environment to Foster Computational Thinking. *Proceedings of the 26th International Conference on Computers in Education*, 1–6. https://arxiv.org/pdf/1608.01392.pdf

Wolber, D. (2011). App Inventor and Real-world Motivation. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 601–606). https://doi.org/10.1145/1953163.1953329

Yaroslavaski, D. (2014). *Relating Lightbot to Programming*. 5. http://light-bot.com/Lightbot_HowDoesLightbotTeachProgramming.pdf